**IST-4-027756 WINNER II****D6.12.2 v1.0*****Report on validation and implementation of WINNER physical layer***

Contractual Date of Delivery to the CEC:	<i>30.11.2007</i>
Actual Date of Delivery to the CEC:	30.11.2007
Author(s):	Marko Leinonen, Marc Laugeois, Sylvie Mayrargue, Huageng Chi, Xiaojia Lu, Mikko Metso
Participant(s):	<i>EBITG, VTT, UOULU, CEA/LETI</i>
Workpackage:	<i>WP6 System Concept</i>
Estimated person months:	<i>67 pm</i>
Security:	PU
Nature:	R
Version:	v1.0
Total number of pages:	55

Abstract:

This is a technical report describing the system overview and technical details of the WINNER Phase II PHY trials.

Keyword list:

PHY trials, adaptive link, SNR estimation, synchronization, LDPC codec

Disclaimer:

Executive Summary

The goal of WINNER is to develop a single ubiquitous radio access system adaptable to a comprehensive range of mobile communication scenarios from short range to wide area. This is based on a single radio access technology with enhanced capabilities compared to existing systems or their evolutions.

WINNER II is a continuation of the WINNER I project, which developed the overall WINNER system concept. WINNER II has developed and optimised this concept towards a detailed system definition. Investigations have taken place within the context of a system view to enable a focused development of a system rather than individual components. In addition limited trials were performed in order to assess some key elements of the WINNER II system.

Within WINNER II, the objectives of the Work Package 6 (System Concept), or WP6, are the following:

- Lead overall WINNER II multi-mode system concept definition by driving the work in the Tasks through concept groups towards a ubiquitous system concept
- Derivation of requirements for the WINNER II concept as targets for the technical evaluations
- Preparation of trials of WINNER II system concept and air interface in Phase III and validation of selected key technologies:

Within WP6 a subtask, T12, was assigned to implement key innovations of the project for small-scale demonstrations and validation. Within the task the PHY trials was to concentrate on the innovations at the lowest layers of radio protocol, and the RRM trials concentrated on the co-operation of the WINNER network with other legacy networks in terms of Radio Resource Management.

Validation strategy in WINNER has been mainly realized with large-scale system-level simulations, based on results from link level simulations and radio channel measurements in the expected operating conditions of the new radio access technology. Since WINNER has been largely a research project, the field has been diverse, but consolidation during Phase II of the project has allowed to build a picture of the expected performance of the system. This is reflected in the final deliverables of Phase II.

Validation strategy within the PHY trials has been to concentrate on the selected key technologies. As results, aspects of implementational complexity has been assessed, and a framework has been build, which allows for measuring and showing the performance of a highly adaptive and highly flexible radio transmission scheme together in a realistic downlink setup. Also, as an important step, the trials have supported the WINNER message and disseminated knowledge in several events throughout the year.

The PHY trials focused on key elements which bring efficiency to single radio link. Among many, an adaptive, flexible, high-performance link, which varies the modulation, code rate, and spatial scheme, on a local sub-bandwidth basis, coupled with an effective forward-error correction scheme, LDPC, were chosen for implementation. To support the work, many other practical estimation algorithms, such as timing estimation (or synchronization), channel estimation, and SNR estimation, are needed to bring a level of realism into the trials concept. Also there are interfaces from the equipment to external components via Ethernet and RF module.

Herein MIMO technologies are regarded as a larger scheme, which is solely defined by the number of input and output points to/from a radio channel, even if the data is correlated. MIMO helps to boost signal-to-noise ratio in difficult radio channel conditions, and can even be used to carry multiple uncorrelated streams of data from radio to another. The technologies as such were not a major scope of the trials, although there was initiative to implement and study fine-grained adaptive MIMO in the PHY trials. The code base largely supports this. However, it was decided to postpone this functionality to the next phase of the project.

Authors

Partner	Name	Phone / Fax / e-mail
EBITG	Marko Leinonen	Phone: +358 40 344 2270 Fax: +358 8 551 4344 e-mail: marko.leinonen@elektrobit.com
CEA-LETI	Marc Laugeois	Phone: +33 (0)4 38 78 53 26 Fax: +33 (0)4 38 78 65 86 e-mail: marc.laugeois@cea.fr
CEA-LETI	Sylvie Mayrargue	Phone: +33 (0)4 38 78 62 42 Fax: +33 (0)4 38 78 51 59 e-mail: sylvie.mayrargue@cea.fr
VTT	Huageng Chi	Phone: +358 40 037 5833 Fax: +358 20 722 2320 e-mail: huageng.chi@vtt.fi
VTT	Mikko Metso	Phone: +358 40 758 7831 Fax: +358 20 722 7053 e-mail: mikko.metso@vtt.fi
UOULU	Xiaojia Lu	Phone: +358 45 111 4388 Fax: +358 8 553 2845 e-mail: xiaojia.lu@ee.oulu.fi

Table of Contents

1	Introduction	6
2	System overview of trials	7
2.1	Equipment details	8
2.1.1	Interfaces overview	8
2.1.1.1	IP interface.....	8
2.1.1.2	RF interface	9
2.1.2	PHY overview.....	12
2.1.3	MAC overview.....	13
2.1.4	Channel models.....	13
3	Link adaptation and resource mapping	15
3.1	Resource mapping	15
3.2	Measurements and estimations	16
3.2.1	SNR measurements	16
4	LDPC channel codec	18
4.1	Introduction	18
4.2	LDPC decoder implementation	18
4.2.1	Decoding algorithm.....	18
4.2.2	Hardware implementation consideration on decoding	18
4.2.3	Decoder architecture	21
4.2.4	Preliminary result.....	26
4.3	LDPC encoder implementation	26
5	Wiener channel estimation.....	28
5.1	Channel estimation	28
5.1.1	Channel estimation in FPGA	28
5.1.2	Channel estimation in Matlab	29
5.2	Wiener filter coefficient calculation on-line.....	29
5.2.1	Selecting necessary calculations for implementation.....	30
5.2.2	Design of basic tools for FPGA to process needed Matlab functions.....	30
5.2.2.1	Trigonometric functions.....	30
5.2.2.2	Matrix inversion.....	32
5.2.3	Selecting parameters for implementation.....	32
5.3	Conclusions	33
6	Synchronization	34
6.1	Synchronisation symbol	34
6.2	Synchronisation algorithm.....	34
6.3	Synchronisation module	36
7	Trials validation.....	38
7.1.1	Deployment scenario.....	38
7.1.2	Transceiver specifications.....	38
7.1.2.1	Forward error correction and modulation	38
7.1.2.2	Multi-carrier technique	39
7.1.2.3	Space-time processing	39
7.1.2.4	Chunk wise processing	40
7.1.2.5	Channel estimation at the receiver	41
7.1.3	Channel model	41
7.1.4	Expected performance.....	42
7.2	System model	42
7.3	Beam domain channel estimation.....	44

7.4	Simulation results	44
8	Conclusion	48
8.1	Trial targets and evaluation	48
8.1.1	Objective 1	48
8.1.2	Objective 2	48
8.1.3	Objective 3	48
9	References.....	50
Appendix A	SNR estimation curves.....	52

1 Introduction

This document describes the WINNER II PHY trials – overall system, technical details, individual key blocks and integration until end of October / November 2007.

A separate report, [WIN2D6124], concentrates on conferences and events attended, and material produced for / from these events, and it also covers, to an extent, integration of the PHY and RRM trials during the project.

The work on the PHY trials is ongoing – during the time of the submission of this report for internal project review, and also after submission to EU. The wordings in the text may at times refer to future work, targeted to the next phase of the project, but this is always explicitly mentioned.

Chapter 2 describes the overall trials system, while chapters 3-7 refer to particular work done by the partners for a point of interest / key element in the L2/L1 protocol levels of WINNER. Chapter 8 presents targets of T12 from the Annex I of Phase II, elaborated to concrete levels. Also an evaluation is carried out, related to these targets.

2 System overview of trials

The main characteristics of the trials system are as follows:

- Full bandwidth signal processing (40/80 MHz), downlink only
- IP packet interface from / to external system, and RF interface to / from radio channel
- Radio channel emulated via external device, with selected WINNER channel models
- IP-traffic and codec data rates in the range of 50-200 Mbps
- FDD-mode SISO-channel support
 - o TDD mode front-end interpolation/decimation filters cannot co-exist with FDD mode filters
 - o MIMO support requires small updates to the datapath, and enhanced estimators. Datapath already supports MIMO
- Estimation algorithms mainly in software, for flexibility

The overall PHY trials system has undergone changes during the course of the project, and it has also changed from event to event, mainly because of integration of code or integration of trials. A baseline overview is given below.

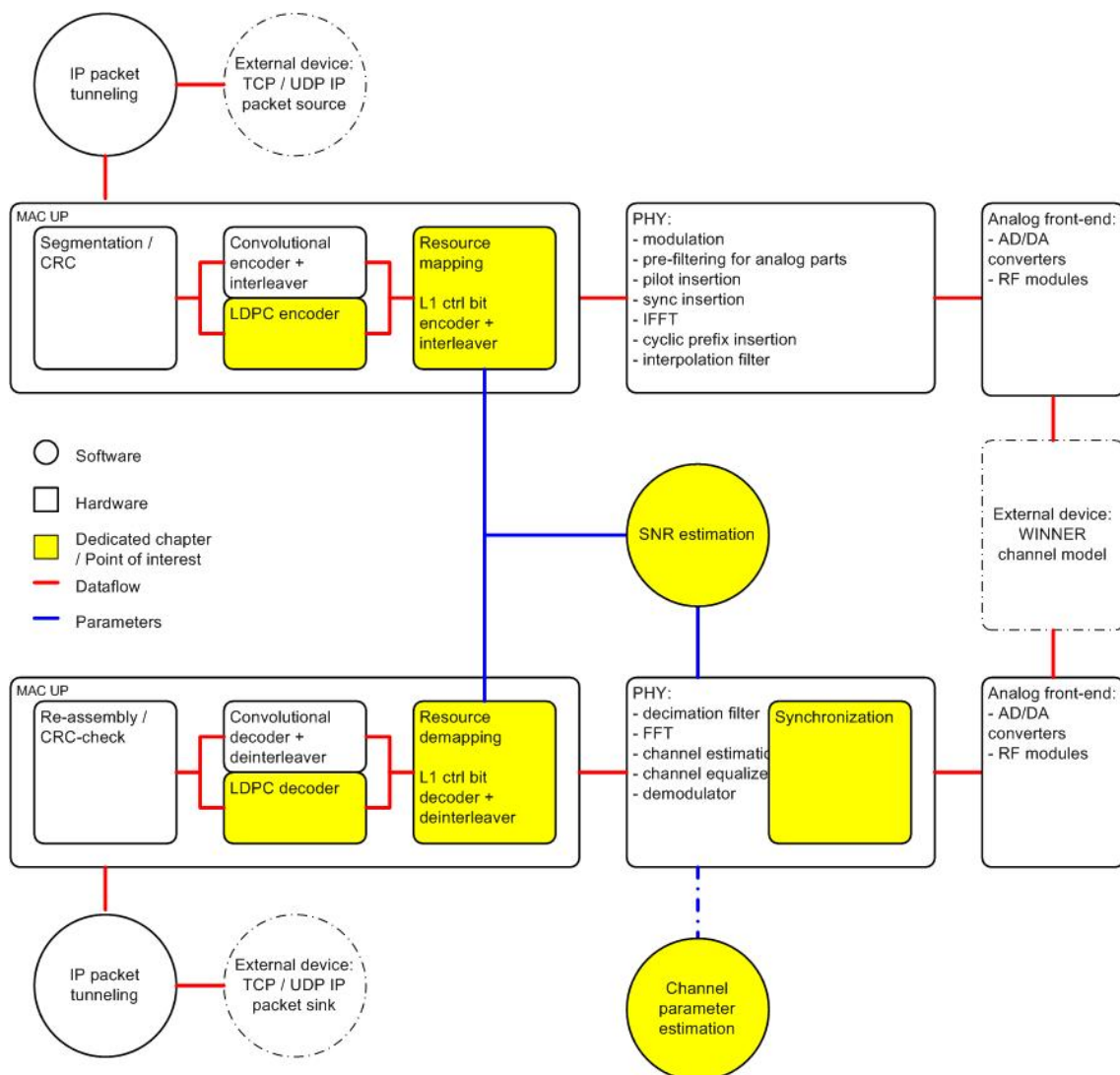


Figure 2-1: System overview and large-scale block diagram

A major overall goal in building the trials setup has been in implementing an adaptive radio link, largely with parameters from WINNER phase I. The work is based on a concept which supports ideas presented in chapter 3.2.2 in [WIN2D222]. Key concepts and building blocks include:

- a LDPC FEC implementation based on parameters investigated in [WIN2D221]
 - o An initial FEC solution is provided with a convolutional code with conventional parameters ($K=7$, $R=1/2$)
 - During the submission of this report, the LDPC codec has not been integrated into the equipment
- idea of decoupling FEC encoding and decoding functions, as well as resulting code block sizes, from the mapping of data onto channel resources, and radio channel resource (chunk) parameters
 - o In principle this allows the use of advanced, iterative coding methods and long codeword sizes regardless of radio channel resource (chunk) allocation to individual user
- fully configurable bandwidth and chunk allocation for 2 simultaneous users / data flows
- adaptive modulation, code rate and MIMO scheme
 - o During the submission of this report, adaptive code rate implementation is waiting for the codec update
 - o During the submission of this report, adaptive MIMO scheme has been postponed for now
- idea of allowing FEC blocks (RTUs) wrap onto 1 or more slots / frames, thus supporting low-bandwidth configuration, and increased link efficiency due to lower amount of zero-padding
 - o It should be mentioned that while the support for a low bandwidth configuration of WINNER air interface is in the general interest of WINNER project, and while this particular implementation may provide efficient use of channel resources, it is not studied in detail, and it requires modifications to L1 control bit implementation. Also, support for any kind of ARQ / H-ARQ algorithm might require changes to this setup

The performance of the system could be measured and validated in many ways, but basically the performance of a single radio link can be simply defined as the achievable throughput, over a SNR-range ($\text{Bit/Hz/s}/(\text{Eb/No})$). Measurements for the existing trials setup could be defined in the following conditions:

- Throughput in AWGN channel: Adaptive vs. non-adaptive link
- Throughput in WINNER channel with low mobility: Adaptive vs. non-adaptive link
- Throughput in WINNER channel: The effect of MMSE-filter update rate

It should be noted that these would not necessarily reflect the true performance of a WINNER radio link, especially considering the lack of H-ARQ, channel predictor and MIMO channel setup in the implementation.

In the chapters 3-6, dedicated points of interest, or major partner contributions to work, seen in “Figure 2-1: System overview and large-scale block diagram”, are detailed. For the rest of the system, a short summary is provided in chapter 2.1.

2.1 Equipment details

2.1.1 Interfaces overview

2.1.1.1 IP interface

The top-level software architecture of the trials system is described in the picture below.

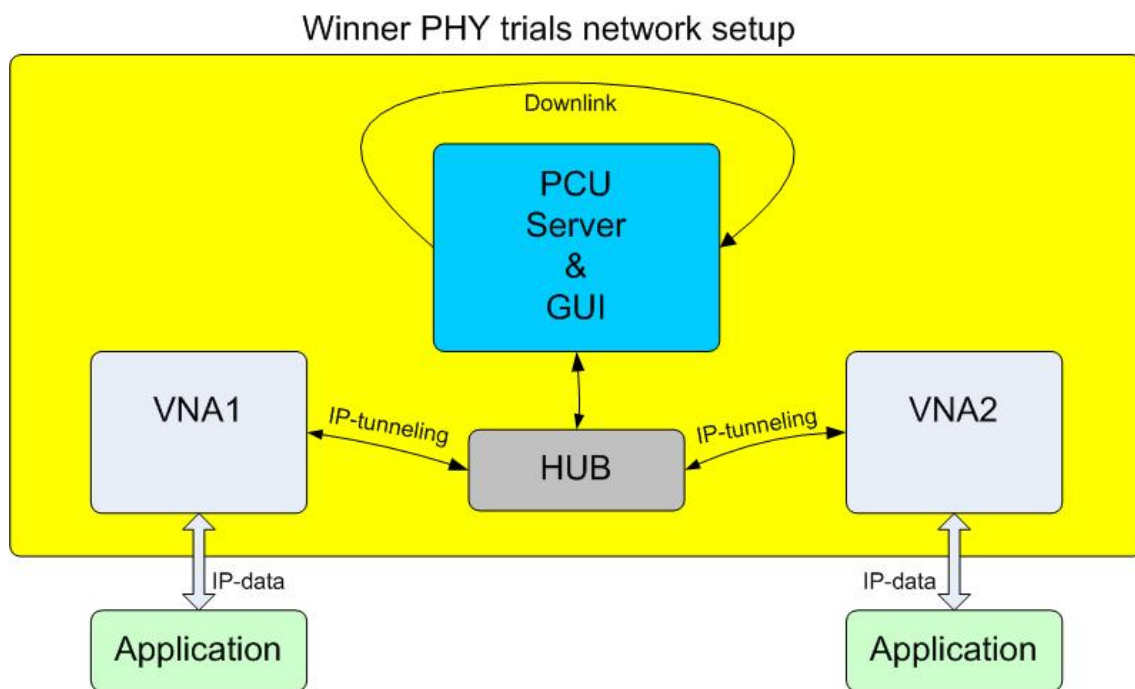


Figure 2-2: Software architecture

The software architecture consists of three parts: Server, GUI and VNA.

- PCU-server is the main program, which handles all general functionality. The main features are: FPGA code loading, IP-data handling and serving of GUI via TCP-sockets. It also collects input data for RRM.
- GUI: Graphic user interface. It prepares the way of tracing the functionality of WINNER PHY-trial during action. The main window gives information about estimated link capacity, user data rates, estimates and adaptation process etc. Also some parameters of WINNER-PHY can be controlled from the GUI
- VNA: The main feature is IP-tunneling, which means that all IP-data directed to certain IP address is forwarded to certain TCP-socket to PCU-server. This makes it possible to handle IP-data in PCU and also makes it easier to change the network topology in the overall trials system.

2.1.1.2 RF interface

The broadband RF module is capable for laboratory and field testing. Wideband RF front-end makes it possible to use the module with various wireless applications.

IF center frequency can be tuned to match different A/D- and D/A-converter interface frequencies (BB to 55MHz). IF filtering is also tuneable.

Configuration of two channels in one module enables small MIMO configurations easily, since the same LO signal can be feeded to both RF chains to avoid phase differences between channels.

Characteristics of the RF interface are as follows:

- RX module:
 - 2-channel broadband RF receiver
 - Frequency bands:
 - low band 300-2700 MHz
 - high band 3500-5800 MHz
 - IF output frequency: 50 MHz (default)
 - IF BW 80 MHz
 - NF 2.5 dB (low band 1 GHz)

- Max gain 75 dB (low band 1 GHz)
- External AGC

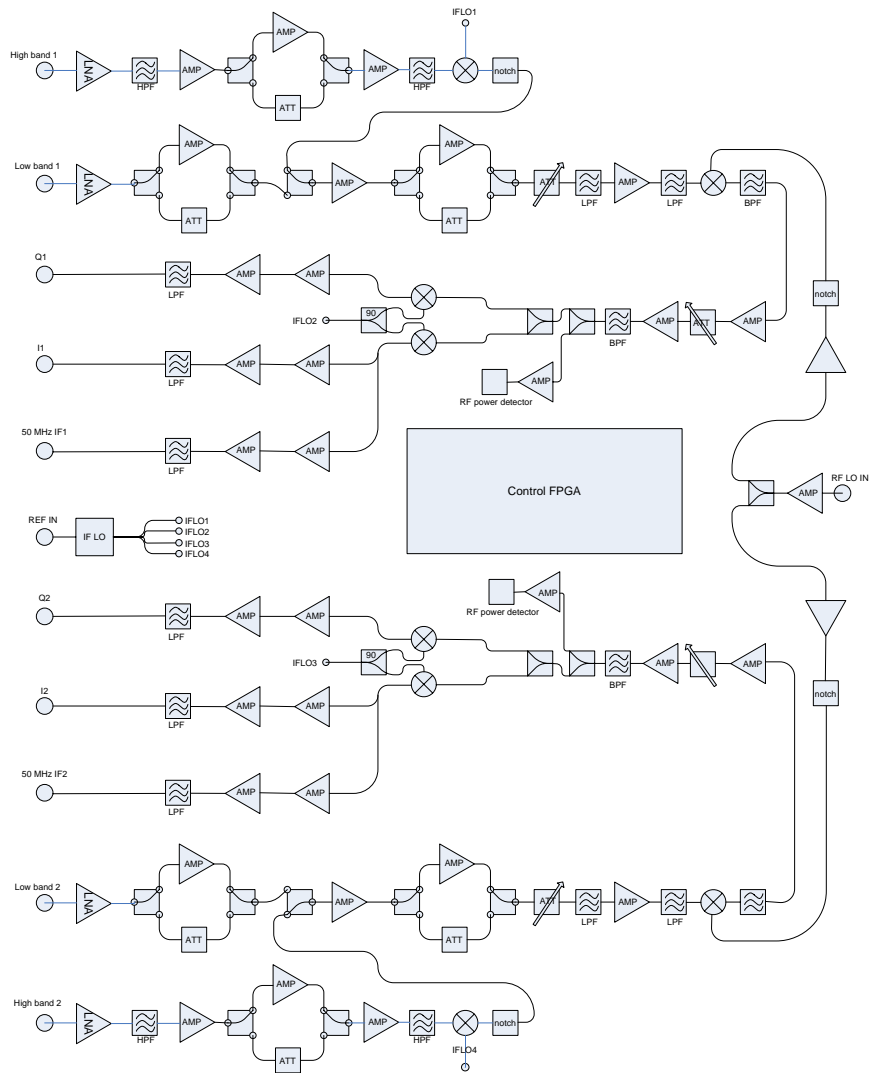


Figure 2-3: RF module RX diagram

- TX module:
 - 2-channel broadband RF transmitter
 - Frequency bands:
 - low band 300-2700 MHz
 - high band 3500-5800 MHz
 - IF input frequency: 50 MHz (default)
 - IF BW 80 MHz
 - output power -40 - 0 dBm
 - power amplifier needed if used for field tests

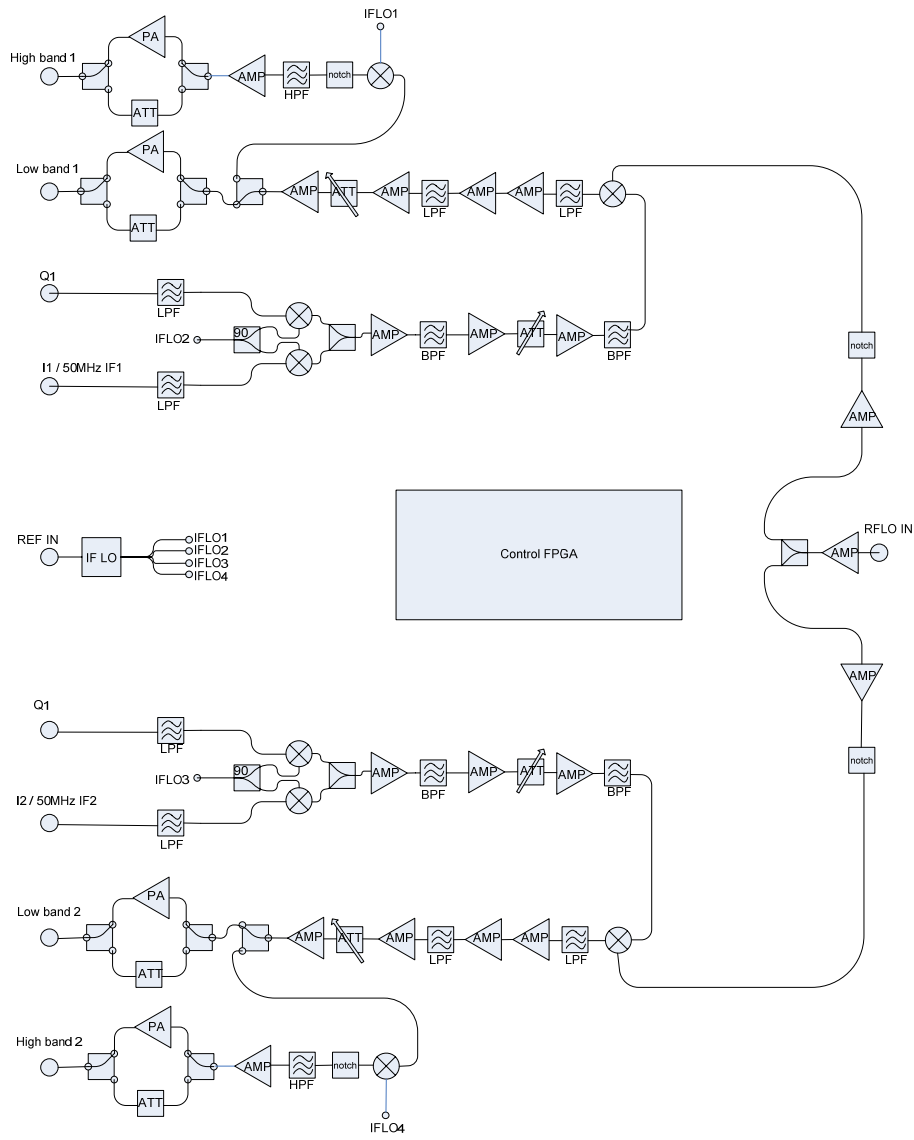


Figure 2-4: RF module TX diagram

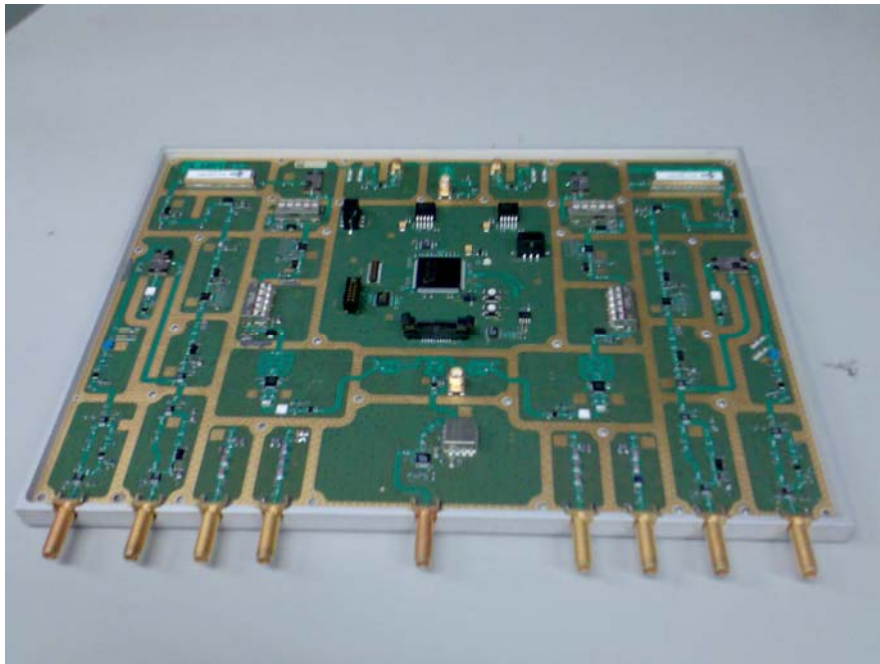


Figure 2-5: RF module

2.1.2 PHY overview

Following picture is an overview of the SISO-mode PHY, which supports both FDD and TDD modes.

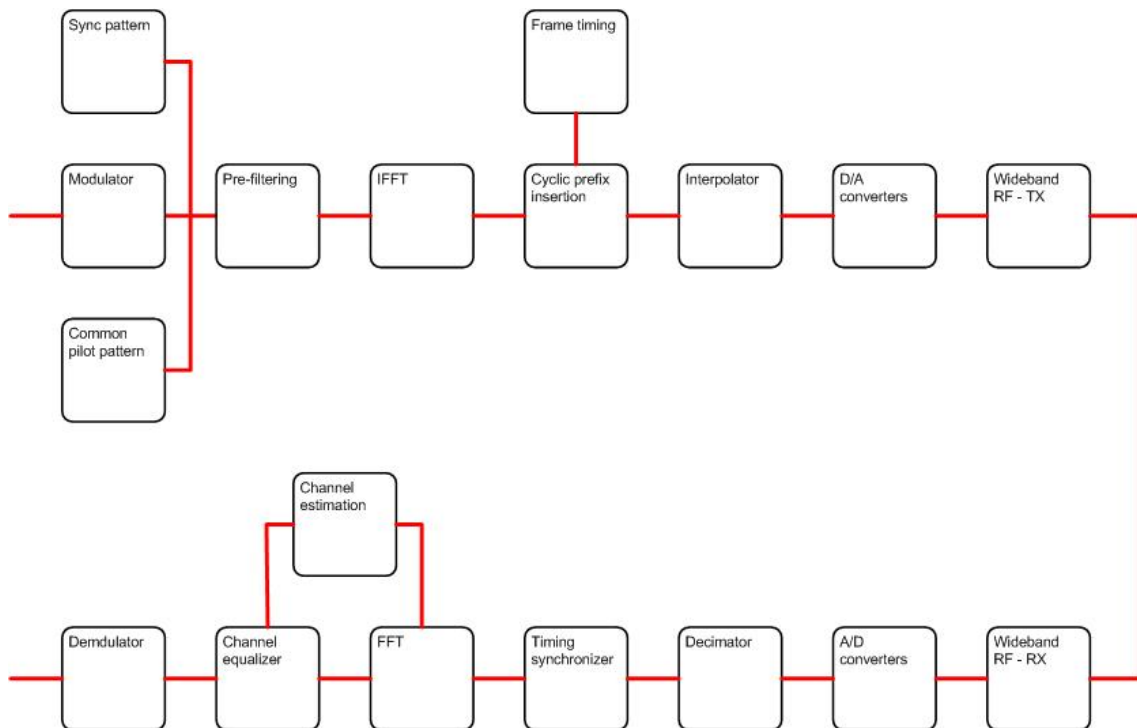


Figure 2-6: PHY overview

Modulator and demodulator support BPSK, QPSK, 16QAM, and 64QAM, but in the setup, BPSK is not used.

Only common pilot pattern is used, as there is no need for dedicated pilot patterns in the demo setup. Pilot pattern is mostly according to guidelines from end of WINNER phase I, some subtle changes (subtle changes in pilot symbol positions) are done to ease the implementation of the channel estimator.

IFFT (and also FFT) are always 2048-point transformations, so that both TDD and FDD modes can be supported easily. This means that the output of FDD data stream from IFFT is already 4x oversampled for 80 MHz bandwidth, and TDD data stream is 2x oversampled for 100 MHz bandwidth.

Interpolator/decimator design does not aim for sharp transitions in the frequency band, as the device is intended to be used with a channel emulator, not real antennas and air interface. Interpolator ratio is 5/2 for FDD-mode, and 2 for TDD-mode. The converters run at clock frequency of 200MHz.

The sync pattern, and consequently the (timing/frame) synchronization has been a simple proprietary synchronization method, until the synchronizer described in chapter 6 has been integrated. The working assumption in the implementation has been that the TX and RX devices are connected to the same clock source, and thus frequency and sample offsets need not be corrected.

Channel estimation uses the PACE method, and interpolates the estimates to data subcarriers. In time domain there is no additional processing, as the channel models used with the trials are intended to be only with low mobility.

2.1.3 MAC overview

The MAC user plane processing at the transmitter side, excluding the resource mapping and functions related to adaptivity, consists of the following functionality:

The arriving IP packets (SDU) are segmented into retransmission units, or RTUs, which is the encoding block for the FEC. The RTU length is dictated by the mother code rate codeword length of the FEC, this is different for convolutional coding and LDPC, and can be changed during synthesis. A 32-bit CRC checksum is added to each RTU, together with a proprietary header for re-assembly. At the moment of writing this, only convolutional code with conventional $K=7$, $R=1/2$ parameters is available, but later a very parametrizable LDPC codec should be also at one's disposal. With convolutional coding, interleaving is performed over each RTU. Finally, resource scheduler buffer stores the codewords, or RTUs, before they are scheduled for transmission. There are 2 more or less independent flows available, which would make it possible to separate different applications with different radio resources, but this has not been pursued further.

The MAC user plane at the receiver side is simply the reverse of operations from the transmitter side.

The data from resource demapper is gathered according to flows and into buffers similar to the ones at the transmitter side. From here, 2 independent flows are first decoded, after which there will be a CRC check. If there are errors, the RTU is not send upwards to the re-assembly.

Re-assembly will check headers from the correctly received RTUs and assembles SDUs based on the information in the headers. If even a single RTU is missing within a SDU, the whole SDU is dismissed. The SDU is also dismissed if it does not qualify against some criteria based on SDU's assumed length, in this case it will be again known that there has been errors in the RTU headers. Those SDUs that can be correctly assembled are sent upwards in the protocol stack.

It should be noted here that it has not been the focus of implementation to study how exactly IP data should be transmitted over a radio link, thus items such as header compression etc do not exist in the trials system.

2.1.4 Channel models

Early on it was decided that real antennas are not to be used with the demo. There were four reasons for selecting this approach:

- Early demonstrations would not have worked with a real propagation environment.
- The use of a channel emulator allows for disseminating info about the WINNER channel models, into which a large amount of research and also implementation work has been consumed.
- The use of the channel emulator allows for more controlled testing environment

- A real propagation environment would have required some extra work for the radio to cope with those propagation effects, which are always present for a real radio, but which can be suppressed for demo equipment, when deemed unnecessary. An example of such an effect is the path loss.

The channel models used at first with a MAC-only setup were snapshots taken from realizations of various scenarios (B1 LOS/NLOS, C1 LOS/NLOS) available in the WINNER channel model. Later, with the integration of the MAC and PHY parts of the trials, more realistic channel models from the same scenarios have been taken into use. In these models an impulse response chain forms a short loop during which the small scale parameters, or ray phases, vary.

3 Link adaptation and resource mapping

Chapter 2 presents all important design ideas for the adaptive link. There are many subtleties in the designing of an adaptive nature for a radio link. The controls must be kept to such a minimum, so that bandwidth is not wasted, but that they are still reliably received, under any circumstances. The same concerns also the use of pilot symbols, which can be used for many estimation tasks. Furthermore, forward-error coding, together with the concepts how the payload, or user data is assembled, coded and divided into packets which need to be reliably transmitted via the radio channel, must also not waste bandwidth.

In the trials design, since the feedback of (slowly changing) estimation results are via cable, no extra control bits are needed in the chunks to inform the receiver about used parameters. This means that only the continuously changing bandwidth of the user data stream (flow) will be reported in the chunk control bits. From these bits, when decoded correctly, the receiver is able to see in what chunks the flow data is mapped, and this allows the user data to be mapped arbitrarily, even into detached chunks. Also, additional information about the concatenated RTUs mapped to adjacent chunks is present, so that in predefined intervals (when the error propability of the control bits gets too large, as estimated from the number of chunks or bits assigned to a flow accumulatively), the mapping scheme is reseted.

This method of allowing RTUs really map freely into the chunks crossing chunk, slot, and frame borders, requires very solid control bit design. It has not been measured yet, but clearly can be seen from the video application used, and also from CRC information, that the current scheme, where QPSK symbols are mapped with data from ($K=7$, $R=1/2$) interleaved, convolutional code, is not always strong enough to carry the control information reliably, with the channel models used. Similar scheme was proposed in an earlier WINNER deliverable. In the future, some tests might be carried out by changing the time basis over which the concatenated RTU mapping is allowed to happen, or then the QPSK modulation might be changed to BPSK, which is, of course, from efficiency point of view, highly undesirable. Control information might be encoded into the pilot symbols also, under certain assumptions, e.g. via differential modulation.

3.1 Resource mapping

Starting from the RSB buffers (the interface between other MAC UP processing and resource mapping), they need to have such a size that at chunk resolution, the radio channel resources can be fully mapped. This might not be a target for a mobile station, but nevertheless, it is not a stringent one.

Data from resource buffer is read symbol-by-symbol by the resource mapper. Since it is assumed that the coding rates, modulations and spatial schemes per chunk are not known during the encoding process, the data residing at RSB buffer is a string of code blocks with mother code rate. Therefore it is the task of resource mapping to perform any code rate adjustments to the data.

In case of LDPC codec, if the code block lengths and RSB buffer bitwidths are selected suitably, the puncturing and depuncturing of the code is merely adjusting the address space read from the buffer. This is the current implementation choice. With convolutional code, one needs a puncturing / depuncturing function, which punctures the data read to the circular buffer form which data is mapped to chunks, and which depunctures the data which is read from the circular buffer, into which data is received from chunk layers. In conjunction with the puncturing scheme, the interleaving function of the mapped RTUs needs to be considered.

Regarding the MIMO case, it is assumed that also spatial layers can be configured with chunk resolution, both in time and frequency, although in real life the time dimension may exhibit more correlated nature. This means that whenever a chunk / chunks possess more than one spatial layer, the coding is a mixture of horizontal and vertical scheme, where mapping always starts with the chunk layer 0 with the lowest frequency, then proceeds to other chunk layers if present, and then moves upwards in frequency to the next chunk.

The user data from the RSB buffer can be freely assigned to an arbitrary combination and number of chunks, in 2 separate flows. There is no real resource scheduler in the system at the moment, but instead a "pre-mapper" which calculates bandwidth needed for each flow during each slot, and assigns chunks accordingly to flows, taking into account code rates, modulations, and usable spatial layers, and the amount of chunks pre-allocated for each flow. The following picture illustrates the capabilities of

mapping, and shows an example how exactly code word bits are mapped onto chunks in the trials implementation.

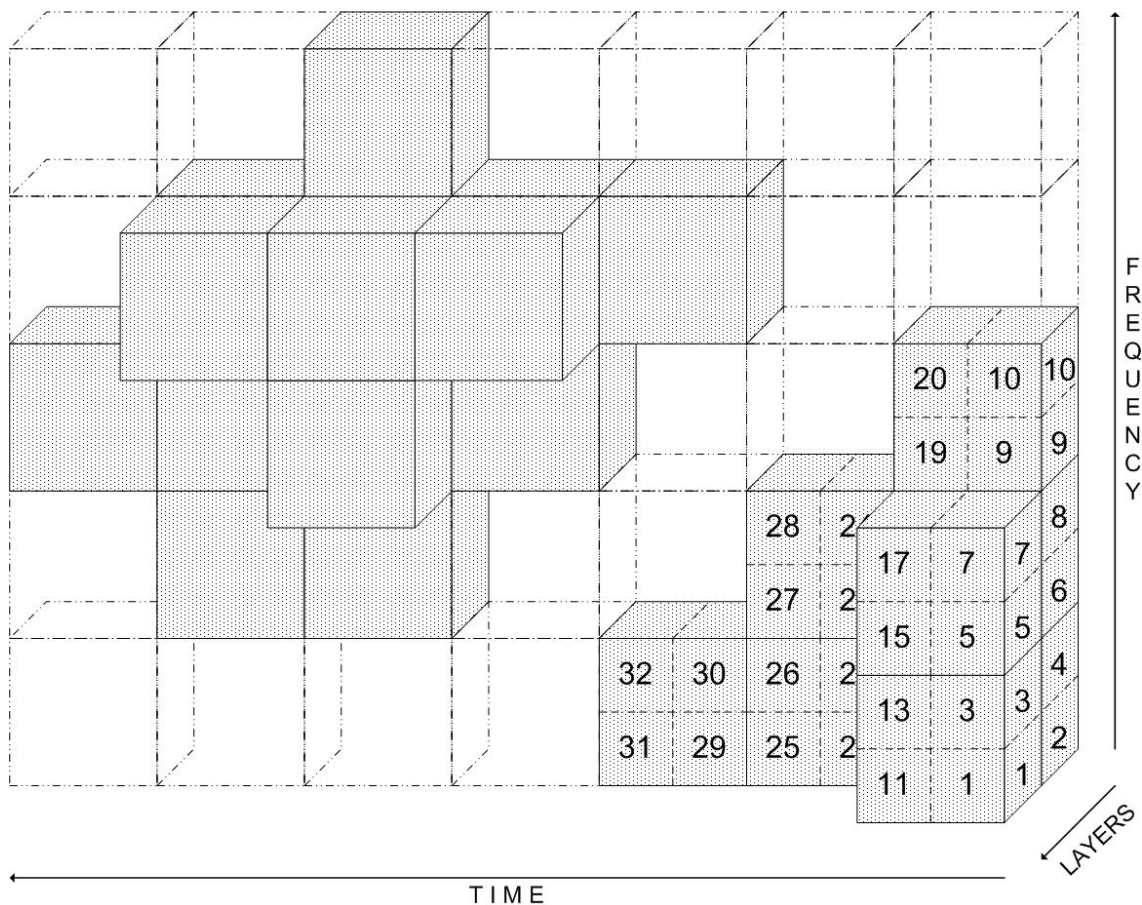


Figure 3-1: Example mapping of user data onto radio channel resources

3.2 Measurements and estimations

3.2.1 SNR measurements

During the project it was shortly studied that, at least for those chunks in which a receiver receives data, and with low mobility, the SNR can be locally estimated for individual chunks, regardless of the modulation used, from the actual data. This would also be possible for those chunks, in which there is data which is not intended for the receiver, but where the absolute SNR level is high enough, the modulation is known, and no user-specific spatial scheme is used.

The estimation is based on measuring norms of signal vectors. From these, noise variance can be easily deducted, with a sufficient sample number. The measurements are done subcarrier-wise, then by averaging over subcarriers within a chunk. For a more realistic version, norms could be measured within a few submatrices in a chunk, due to the possible variation of the real SNR in frequency (small channel coherence bandwidth, or high frequency-selectivity), and in time (small coherence time, or high mobility), or then some kind of channel compensation would have to be arranged..

Appendix A shows a few simulations from a more advanced method of estimation. So far these simulations have been run in AWGN channel, and it is estimated that introducing a real channel will worsen the results somewhat.

The scheme employs 2 pilot symbols (normalized amplitude 1), 10 control symbols (QPSK) and 80 data symbols (16QAM or 64QAM). The channel is estimated from the pilots, and the SNR is estimated after that from the control symbols (QPSK results), and also from the data symbols (16QAM/64QAM results), since the large number of them should help reducing the errors in the estimation. In the first batch of

results, it can be seen that clearly there is a large bias and variation in the 16QAM and 64QAM results. By having very simple iterations on the control symbols, the results are better in the second batch of results, and by having some calculation iterations on the data symbols, it can be seen that even the 64QAM allows for quite accurate measuring of the SNR, even in the region where it would not normally be used, albeit with some bias in the result. It should be noticed that the channel estimates themselves are also improved in the process, and that the iterative method does not depend on the channel codec or the modulation alphabet.

These results should mean that accurate estimations of the channel quality are possible, and they are essential for very fine-grained link adaptation, into which category code rate adaptivity falls. Without estimation accuracy, link capacity decreases.

The SNR estimator was implemented in software. There is one drawback with the use of software-based estimators, which is that estimations take time, and only very low mobility can be demonstrated. Also, the actual writing of the parameters into the hardware takes some time, which sometimes causes different parts of the setup to use different parameters during the time of adaptation, creating errors in the data stream. This behaviour can be alleviated with longer timebases for the change of parameters, or by synchronizing the units after the information has been written to all necessary locations in the equipment.

4 LDPC channel codec

- section 4.1 presents a short summary
- section 4.2 presents LDPC decoder
- section 4.3 presents LDPC encoder

4.1 Introduction

This chapter presents the FPGA implementation of low density parity check (LDPC) forward error correction channel codec. Decoder implementation and encoder implementation are presented in section 4.2 and 4.3 respectively. The following list is a brief summary of the implementation.

- Code design:
 - The parity check matrices, which define WINNER LDPC codes, are given in [WIN2D221]. The codes belong to irregular block LDPC.
 - [WIN2D221] defines 3 base matrices, whose dimensions are 24x48, 16x48, and 12x48, respectively for code rate $\frac{1}{2}$, $\frac{2}{3}$ and $\frac{3}{4}$. Modulo expansion is used to expand base matrices to low density parity check matrices that directly characterize LDPC codes. The 5 block sizes for each rate are 12, 24, 36, 48 and 96, corresponding to codeword length 576, 1152, 1728, 2304, and 4608. Thus 15 codes are defined in [WIN2D221]
 - Note: a matrix defined in [WIN2D221] has a *tiny* difference from that used in FPGA implementation. It's in the last column of base matrix for rate 1/2. Implementation started before [WIN2D221] was released, and the implementation was not updated.
- The codec implementation targeted device is Xilinx Virtex II Pro XC2VP50-5FF1152.
- The implementation is capable of real time configuration to one of 15 codes, i.e., code rates and codeword lengths are run-time configurable.
- Encoding algorithm is based on [RU01].
- Decoding:
 - Implemented algorithm is min-sum correction (scaling factor 0.8), which is an approximation to sum-product algorithm (SPA), also known as belief propagation algorithm (BPA).
 - Implemented scheduling is horizontal shuffling, also known as layered scheduling.
 - Check node update processing is serial and pipelined. Messages to and from check nodes are out-of-order, in order to increase throughput.
 - Parallelism factor is 96, which is the maximum of block size.
 - Dual buffers are used to increase throughput.

4.2 LDPC decoder implementation

- section 4.2.1 presents algorithm
- section 4.2.2 presents hardware implementation issues in general
- section 4.2.3 presents implemented decoder architecture
- section 4.2.4 presents info of preliminary result

4.2.1 Decoding algorithm

For more information on LDPC decoding algorithm, refer to [WIN2D221], as well as WINNER phase I deliverables [WIN1D21], [WIN1D23] and [WIN1D210].

4.2.2 Hardware implementation consideration on decoding

General objectives of hardware implementation task

While performance of LDPC codes is largely determined by code design task, the hardware implementation task mainly concerns about:

- Mapping selected decoding algorithm to hardware structures
- Feasibility to fit a hardware structure to targeted hardware device (Xilinx Virtex II Pro)
- Guarantee a reasonable decoding throughput at a moderate clock frequency
- A particular issue to LDPC codec implementation is configuration, that is, capability to select one of 15 codes in real time. Overhead due to configuration should be minimized.
- More advanced objectives can be:
 - utilize targeted device efficiently
 - high speed implementation (in terms of clock frequency)

The challenge of implementing a LDPC decoder comes from simultaneously considering those issues above.

General considerations for LDPC decoder hardware implementation

LDPC uses iterative decoding (message passing). In contrast to other iterative decoding methods, such as BCJR for turbo codes, LDPC code features an inherent parallelizable decoder structure, which is well explained by the standard sum-product decoding algorithm (SPA) on Tanner graph describing a LDPC code. Fully parallelized decoder architecture can be obtained by direct mapping a Tanner graph to hardware. In this case a decoder is a connection of large number of processing nodes of two types, one is variable update node, and the other type is check update node. This solution results in high throughput, at the cost of large processing scale. Each node itself is simple, however, number of nodes is large, and total complexity is high. Routing congestion is severe due to large number of wires connecting nodes. Flexibility of real time configuration is hard to implement in such a solution. On the contrary, fully serialized architecture is suitable for application based on DSP processors, where a single processor executes codes on behalf of each processing node in time multiplexed manner, and interleaving structure (connection of nodes) is implemented by memory operations. Full serial architecture avoids above problems, at the cost of low throughput. This discussion demonstrates the flexibility of LDPC decoder implementation. Some key aspects to consider during LDPC decoder implementation are identified, and listed below.

- **Parallelism factor.** High parallelization produces high throughput, often at the cost of higher gate counts, larger amount of memory, and severer routing congestion, which can limit the factor. For example, a direct mapping of base model matrices implies flooding scheduling architecture, which needs 189 dual-port block-RAM modules (189 is the maximum number of block edges). Parallelism factor is small, because increasing the fact requires more memory modules that a chip cannot offer.
- **Memory requirement.** Candidate architectures have different on-chip memory requirements.
 - For example, a candidate architecture using flooding scheduling for WINNER defined LDPC may use about 189 dual port memory modules to hold exchanged messages. Memory requirement is large, while memory usage rate is low.
 - In a second example, improvement can be made by using so called message broadcasting, which is applicable if check node update uses min-sum algorithm. Value reuse property of min-sum's check node updating enables memory compression. Each check node saves and broadcasts compressed messages to its neighbour variable nodes. A variable node makes a sum and broadcasts the sum to neighbouring check nodes. Memory reduction is achieved, at the cost of more complex control logic.
 - In a third example, layered scheduling with serial check node updates enable use of only one dual-port memory module to hold variable messages. Less number of on-chip FPGA ram blocks is needed due to relatively narrower memory bandwidth compared with previous examples. The disadvantage is low throughput because decoding is performed layer by layer in serial, and blocks of a layer are read in serial. As memory requirement for variable sum is reduced, additional memory is needed for each check node to save check-to-variable messages, which has 189 entries (189 is number of maximum block edge number defined in WINNER LDPC codes).

- Note that, besides storage for messages, other parameter requires memories too. For example, memories are needed to save code structure, which are essentially the base model matrices. Shifting values require memory storage too.
- **Decoding algorithm.** There are several variants of decoding algorithm. Sum product algorithm, also known as belief propagation, is considered the standard. Min-sum with scaling factor approximation is well known and adopted by WINNER implementation. [WIN2D221] narrows down to two options, namely, flooding scheduling and layered scheduling. For each one, there are more than one candidate architectures, which are different in aspects like memory requirement, wiring, logic scale, etc.
- **Throughput.** At first glance, flooding scheduling needs more iterations to converge for low SNR, which implies lower throughput than layered scheduling. However, layered decoding needs more operations to complete one iteration. For serial check node updates, it needs even more clock cycles to pass messages to a node. Serial check update node is simple, which is good to increase parallelism factor. But the factor for flooding scheduling given the targeted device is hard to increase. Flooding scheduling favours parallel check node update, and their latency should be minimized. The configuration requirement makes the node costly, because there are nodes of different degrees, but only a fraction of them is active for a given code rate.
- **Targeted hardware.** As a general rule, targeted hardware device affects architecture selection. For example, while multiplexer is cheap in ASIC, it's not so cheap in LUT based FPGA. Some FPGA has large number of smaller on-chip memories, while others have less number of larger memories, different hardware features favour decoder architectures slightly different.
- **Inter leaver.** The essence of interleaving is to introduce randomness. Unlike others such as turbo code, LDPC has no explicit inter-leaver. The randomness of LDPC is encoded into the connections of nodes, i.e., the edges of Tanner graph reflect randomness. For a (n,k) code, the edges of a Tanner graph can be viewed as a $(n$ -input, n -output) inter-leaver. The inter leaver is costly due to its dimension, and particularly with objective of runtime configuration, which requires an inter-leaver to configure to different topologies, and select the right subset of variable nodes and check nodes to connect. For FPGA, inter-leaver also consumes large amount of routing resources, adds to data path latency, and decreases throughput. The problem is partly avoided by layered decoding with serial check node update, where inter connection is reflected by memory read and write. A well known problem in layered decoding is memory access conflict. It occurs when a layer decoder tries to read a block, which is not yet updated by previous layer. Some codes consider this during code design such that rows of a base matrix can be permuted to reduce inter-layer dependency. This problem is not emphasized in WINNER defined LDPC. Another walk around is to apply out-of-order read and write, that is, a layer decoder writes back to variable sum memory those blocks needed by next layer first; and a layer decoder reads such blocks last. This can greatly reduce stall states, which is otherwise needed.
- **Shifter.** Depending on architectures, one or more circular shifters may be needed. For example, layered decoding with parallel check node updates needs 16 shifters, where 16 is the maximum check node degree defined in WINNER LDPC. Serial check node update can reduce the number to one. Flooding architecture may avoid use of shifter if decoding is not parallelized within a block. Logarithm shifter is commonly used. Real time configuration adds additional requirement on shifter, which requires a shifter to perform circular shifting on different ranges of a 96 elements-long vector. For example, for block size of 96, the shifter performs circular shift on all 96 elements of a vector, while for block size 48, the shifter performs circular shifting on first 48 elements of 96 element-long vector.
- **Check node update.** Decoding tasks can be divided to two kinds, message passing and message update. Two types of message update nodes are variable node update, and check node update. Check node update processing is more complex than variable node updates, and receives more attention. The implementation selects min-sum correction for check node update. Parallel node is favoured in flooding scheduling. Serial node can be used in layered scheduling. Serial node is simple, at the cost of longer latency, but it enables higher parallelism factor. Serial node is also flexible, which can compute nodes with different degrees efficiently. Parallel node has fixed number of input ports and output ports, and the situation becomes complex because there are 6 degrees for all the 3 rates, i.e., degree 2, 3, 10, 11, 15 and 16.

- **Runtime Configuration.** Run time configuration to select one of 15 codes adds flexibility to LDPC codes, and meanwhile adds overhead to hardware implementation. The overhead is different to variants of architectures.
- **Relation.** This list identifies some aspects for decoder implementation. Those aspects are related. A good decision in one aspect might have negative effect on others. An implementation needs find a balance among aspects of interest, provided system requirement and hardware constraints.

Selected architecture to implement

Several candidate architectures have been evaluated, including

- Flooding scheduling
- Flooding with broadcasting
- Layered scheduling with parallel check node update
- And layered scheduling with serial check node update.

For each of them, those aspects listed in the preceding list are studied in order to find a good trade-off.

The FPGA implementation uses layered scheduling with serial check node update. This architecture requires less memory and routing resource. It uses simple check update nodes, thus enables high parallelism factor. Check node update is pipelined to increase throughput. Out-of-order read and write are applied to check node update to reduce interlayer dependency. Dual buffer is used for IO task to further increase throughput. Variable sum memory depth is 48, corresponding to 48 blocks of a codeword. Memory bandwidth is 96 messages-wide, reaching the maximum parallelism factor.

4.2.3 Decoder architecture

This section outlines implemented decoder architecture.

Dataflow view

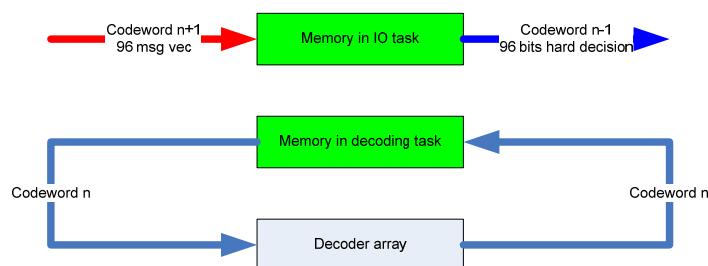


Figure 4-1 Dataflow view

Figure 4-1 outlines the data flow. Two memories are identical. A codeword has 48 blocks, depending on codeword length, block size can be 12, 24, 36, 48 and 96. A message is represented by 5 bits two's complement number. A block of 96 messages is saved in one memory entry, requiring $96 \times 5 = 480$ bits-wide memory bandwidth. The memory modules are 48 entries deep, corresponding to 48 blocks of a codeword.

Decoder starts from idle state after system reset. Incoming codeword arrives at decoder block by block continuously, taking 48 clock cycles. When a memory is filled with a codeword, it's switched to decoding pipeline, and decoding starts. Decoder has internally 96 serial check update nodes, thus decoding pipeline can take in one complete block at a clock cycle. Decoding starts from 1st layer. All blocks checked by 1st layer are read into decoding pipeline (in serial), and then the 2nd stage of pipelined check node starts to write updated message back to memory, meanwhile, blocks for next layer are being read into 1st stage of decoding pipeline. In such a manner decoding is performed layer by layer, iteration by iteration. Decoding terminates when hard decisions converge, or maximum number of iteration is reached.

As decoding pipeline is active, the other memory is able to accept next codeword. When decoding is done, two memories are swapped, and decoding of the newly saved codeword starts immediately.

Decoded result in the memory swapped to IO mode is read out and sent to decoder output port. After the decoding result is sent out, the memory in IO mode can accept the next incoming code word for decoding.

Layer dependency

A problem in layered scheduling is interlayer dependency. Out-of-order variable sum memory read and write can solve the problem partly. When blocks of a layer are written back to memory from check nodes, those blocks that are read by the next layer are written to memory first. When blocks of a layer are read from memory, those blocks that are supposed to be updated by preceding layer are read in the last. The order of read and write, i.e. arrays of memory address are saved in ROM modules. An address array for one code rate is different from another code rate. Memory conflict may occur even out of order memory access is used. If it happens, read operation must be paused until the memory is updated. Hardware blocks of decoding pipeline communicate with each other using simple handshake, in order to cope with stall states.

Configuration

Code rate and code length are run time configurable. Code rate configuration is done by selecting the correct memory access addresses array to read and write variable sum memory. Code length configuration is related to shift values. Each length has a set of shift values. The values can be derived in run time from those values given in the base model matrices. In this trial implementation, the values are pre-calculated and stored in ROM modules. This is feasible due to large number of on-chip memories, and it simplifies implementation. A shift array stores shift values for a code with specific length and rate. The length of the array is 184, 185 or 189, which are the number of nonzero blocks in the base model matrices for three rates.

Interface view

Figure 4-2 presents interface of the decoder.

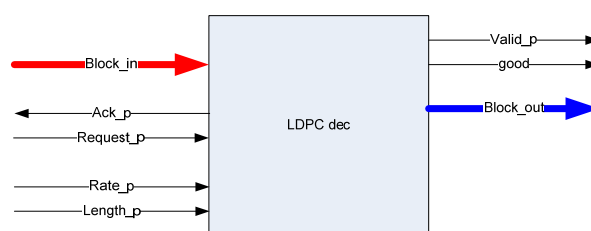


Figure 4-2 Interface

Decoder user sends request pulse to decoder, meanwhile providing code rate and codeword length information. Decoder replies acknowledgement pulse in the following clock cycle if it can accept a codeword. Starting from the next clock cycle after ACK pulse, decoder user should send 48 blocks of a codeword using 48 consecutive clock cycles. Decoder does not reply ACK pulse if it's busy, and decoder user can continue request or not.

When decoder intends to send out decoded result, it first makes a 'valid' pulse to notify the user of decoder, and decoded result will be sent out starting from the next clock cycle, using 48 clock cycles. If decoding was converged, 'good' is set to '1'.

Top level architecture

Top level architecture of the decoder is shown in Figure 4-3.

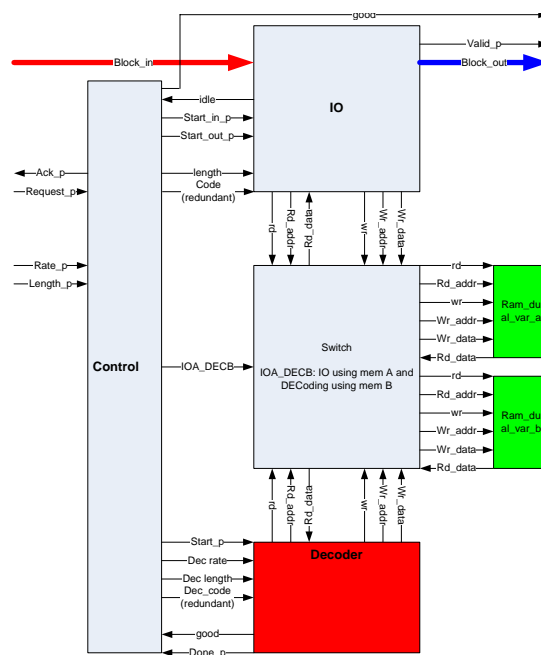


Figure 4-3 Top level architecture

Two green blocks represent two dual port memory modules, when one is in decoding mode, another can be in IO mode (or idle). The red block is the decoder core performing LDPC decoding. IO block manages input and output of code words. Switching box swaps memories for IO and decoder core blocks. Those sub blocks are controlled by a control block, which sends start pulse to IO and decoder core, and wait for response and take proper actions.

Data flow view of decoder core

Figure 4-4 outlines the data flow of decoder core.

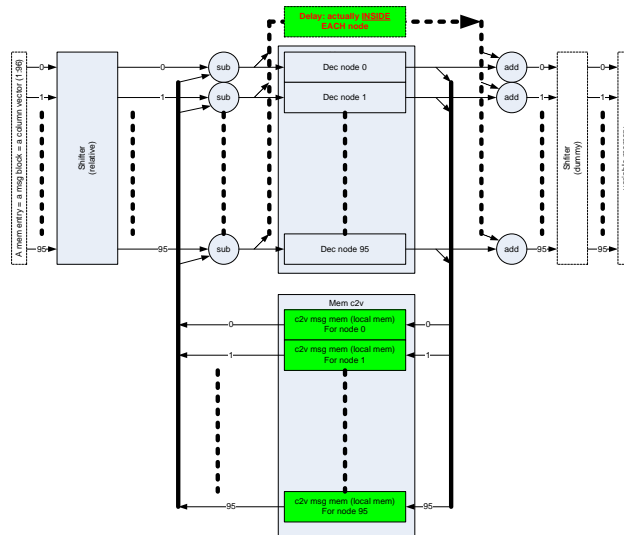


Figure 4-4 data flow of decoder core

A block of maximum 96 messages is read out of variable sum memory and arrives at shifter. Not all of 96 messages are valid, depending on the codeword length. Invalid messages enters decoder pipeline and get written back to memory as well, but their values should be ignored by decoder user. Shifter block performs circular shift on the block of messages, which can be viewed as a vector of 96 values. Depending on the codeword length, the circular shift is performed on the first 12, 24, 36, 48, or all 96 elements of the vector. Shifting values are derived from those values given in the base model matrices. Shifting values can be computed in run time, but this trial prototyping implementation saves pre-calculated shifting values to ROM modules, and those values are fed to shifter in run time. The values in

base model matrices are absolute shifting values, they are supposed to direct shifter. Shifted blocks enter check node update array, and the output of check update node array are shifted back before the messages are saved back to variable sum memory. This solution requires two shifters in decoding pipeline, which cost more logic and increase latency. The implementation omits the 2nd shifter, so that a block is not shifted back when it's saved to variable sum memory. When the block is used next time in decoding, the shifter should make an incremental shift, in stead of absolute shifting. In other words, the shifter performs relative shifting only.

A subtract array is needed to reproduce variable-to-check messages. Values from variable sum memory are minuends of the subtract operation, values from local memory of each check update node are subtrahends of the subtract operation. The subtrahends are check-to-variable messages which are computed in previous iteration where the block is involved.

Output of subtract array enters check node array. Each node performs min-sum correction decoding algorithm. The pipelined check node has two stages, when the second stage outputs updated check-to-variable message, the first stage can accept message block of the next layer.

The output of check node array is saved in memory, and they are to be used in the next iteration. Meanwhile, the output also enters adder array, which add newly updated check-to-variable messages to delayed version variable-to-check messages, to produce variable sum messages.

The output of adder array is written back to variable sum memory without going through inverse shifter, which is omitted in the implementation.

Architecture view of decoder core

Figure 4-5 presents architecture view of decoder core.

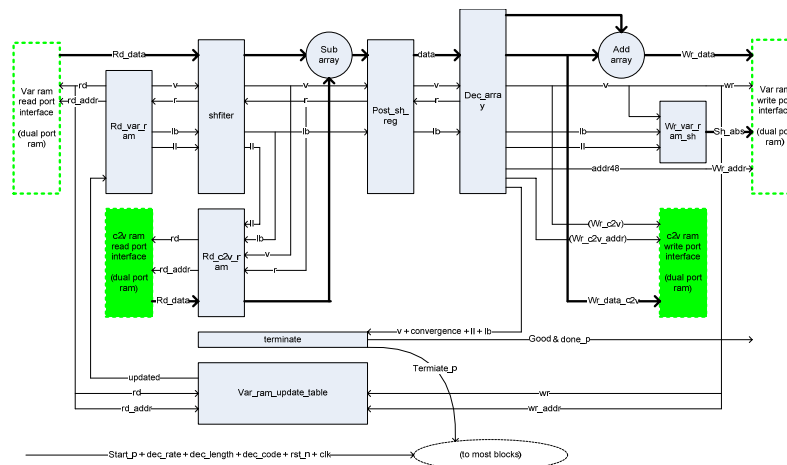


Figure 4-5 Architecture view of decoder core

The architecture view reviews actual hardware entities and connections of the decoder core. The figure is similar to that data flow view.

- **rd_var_ram** is a block that fetches message block from variable sum memory. It checks first if an addressed block has been updated. If not updated, it waits until it's updated. The data out of variable sum memory is routed to shifter, using valid-ready handshake. lb and ll, meaning last-block and last-layer, are two flag signals. rd_var_ram has internal ROM module to save addresses for memory read operation. There are 3 set of address, the size of each set is 184, 185 and 189 respectively for 3 rates. Code rate configuration is done by selecting the right set of addresses.
- **Shifter** performs circular shifting on part, or all, of a message block. Shifter has internal ROM modules storing shifting values. The values are relative shifting values, because inverse shifter is omitted. There are 15 sets of shifting values, corresponding to 15 codes of different code rates and word lengths. Output of shifter goes through subtract array and get saved in register (post_sh_reg).
- **Subtract array** reproduces variable-to-check messages by subtracting check-to-variable messages produced in last iteration from variable sum messages out of shifter. The check-to-variable messages are read out from memory.

- **rd_c2v_ram** reads check-to-variable messages out of check-to-variable dual port RAM, and routes data to subtract array. All check-to-variable updated messages are saved in this memory.
- **post_sh_reg** is a register inserted in the pipeline
- **dec_array** contains 96 identical pipelined check update nodes, implementing min-sum correction algorithm.
- **add_array** adds updated check-to-variable messages to delayed version variable-to-check messages to produce variable sums.
- **valid-ready** form a simple handshake rule used by several sub blocks. Data receiving block asserts ready when it can accept data in the current clock cycle. Data sending block asserts valid when data is valid at its output.
- **wr_var_ram_sh** writes absolute shift values attached to each block to variable sum memory. Each entry of variable sum memory contains 96 messages fields and one field for shifting value attached to the block. The value reflects the absolute shift applied to the message block in a memory entry. This value is used in IO block for output message block.
- **terminate** stops decoding on conditions that either decoding is converged or maximum iteration limit is reached.
- **var_ram_update_table** keeps record of each message block in variable sum memory. When a block is read out and sent to decoding pipeline, the flag of that block is clear to zero, meaning not updated. When a message block is written back to variable sum memory, the flag associated with that block is set to one, meaning updated.

Shifter

A shifter is needed to perform circular shift on a 96-element-long vector. Vector elements are indexed from 1 to 96 (top-down), the 5 ranges that circular shift operation performed are

- 1-12, in case of block size 12
- 1-24, in case of block size 24
- 1-36, in case of block size 36
- 1-48, in case of block size 48
- 1-96, in case of block size 96

For circular shifting on a fixed length vector, logarithmic shifter is commonly used, which perform a circular shift ranging 0 to S-1 using $\log_2(S)$ levels of 2:1 multiplexers. To solve circular shifting on multiple lengths, a two stage shifter is used. First, a linear vector indexed from 1 to 96 is rearranged and viewed as a 12x8 matrix, as shown in following table.

1	13	25	37	49	61	73	85
2	14	26	38	50	62	74	86
3	15	27	39	51	63	75	87
4	16	28	40	52	64	76	88
5	17	29	41	53	65	77	89
6	18	30	42	54	66	78	90
7	19	31	43	55	67	79	91
8	20	32	44	56	68	80	92
9	21	33	45	57	69	81	93
10	22	34	46	58	70	82	94
11	23	35	47	59	71	83	95
12	24	36	48	60	72	84	96

The shifting operation is best explained with an example. In the example, the valid range is 1-48, (shaded) corresponding to block size 48 (codeword length 2304). Shifting value is 42, meaning to shift the vector [1...48] to [43, 44 ... 48, 1 ... 42], while vector elements in position 49 to 96 are insignificant. The first step is horizontal circular shift performed on each row. Shifting step for rows 7 through 12 is 4, for rows 1 through 6 is 4+1=5, equivalent to -1. Note that the shift range that the circular shifting performed on row is 1 through 4 (the 4 left most elements of a row vector). The rest of a row (the rightmost 4 elements) is insignificant, because block size is 48, and the first 48 elements are contained in the first 4 columns of

the table. The second step is vertical shifting. In this example, each column is circular shifted up by step 6. After that, the first 48 positions of the table contain the wanted circular shifted elements.

Check node update

Check node update details are presented in Figure 4-6.

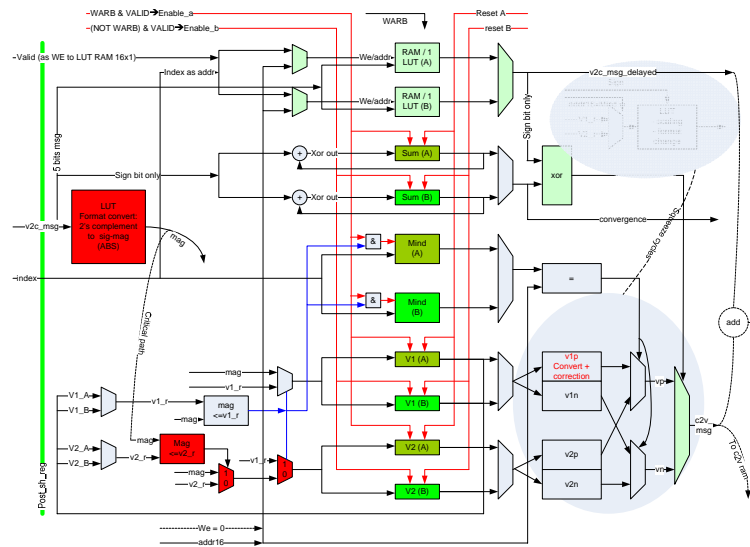


Figure 4-6 check node update

In the middle are two sets of registers, labelled with A and B. The register divides check node to 2 stages of a pipeline. As input stage writes a set of register, output stage can read another set, which contain values readily written by input stage, and sends data out of check node.

Min-sum algorithm favours sign-magnitude format to represent values, a look up table in the left does conversion. Signs and magnitudes are handled separately in min-sum algorithm. Magnitude output has a useful property that the output contains at most 2 distinctive values. This property allows compressing the output to 3 values, v1, v2 and ind. If the input magnitudes have a unique minimal, denote it as m1, and its value v1, its position ind. v2 is the minimal among the rest of input magnitudes excluding v1. As magnitudes enters check node in serial, the check node keep record of two smallest values and the index of m1. In output stage, check node receives an index for read operation, and checks if it equals to ind, it takes v2 as magnitude for output if they are equal, takes v1 otherwise. This method also works if the input magnitudes have multiple minima.

4.2.4 Preliminary result

A preliminary synthesis using Xilinx ISE 9.1i was made targeting Xilinx Virtex II Pro FPGA (2vp50ff1152-5). The implementation consumes 55% slice resource, 50% LUT resource, 8% slice flip flops, achieves 100MHz clock frequency. On chip memories are treated as black boxes in synthesis. 43 block memories are used, corresponding to 19% block memory resource. Current implementation uses 5 bits two's complement number representation. Increasing number of bits can improve decoding performance, meanwhile increases memory requirement, logic size, and decreases clock speed. Decoding throughput is not fixed, because of variant number of iteration (depending on channel condition), and it also depends on selected code rate and word length. For successful decoding in worst channel condition, decoding iteration reaches the maximum number of 20 iterations. In general, considering this implementation architecture, higher block size implies higher throughput. And at each block size, higher code rate implies higher throughput. In the worst case (20 iterations), throughput at 100MHz clock frequency is estimated to range from 10 to 70 Mbps (user data, i.e., decoded data bits per second), depending on code rate and word length. Note that the numbers obtained from preliminary synthesis are subject to change during ongoing refinement of the work.

4.3 LDPC encoder implementation

LDPC encoder architecture is drafted by H. Chi (VTT), and improved, implemented, tested, and finalized by Y. Zhang (VTT). Refer to Zhang's report [YZhang] for details. The report is located on BSCW server:

- document name: WINNER_LDPC_ENCODER_REPORT.doc

- document link: <https://bscw.eurescom.de/bscw/bscw.cgi/0/367950>
- The document can be found in folder named 'reference', which is located in the same directory on BSCW server where this deliverable resides.

5 Wiener channel estimation

A need for OFDM channel estimator (CE) initiated an implementation work towards a functional design in an FPGA. Initially this was not seen as a substantially demanding task but as this was reviewed deep enough, more and more unknown issues came up.

Lots of related literature exists to help the designers get acquainted with the subject, for example on digital signal processing and general filter theory ([The92], [Hayes96], [Haykin02], [Vas06], [IJ02] and [RG75]), as well as channel estimation and Wiener filter theory ([SSK06], [HKR1], [HKR2], [Auer05] and [CIS+06]).

The original task was to implement a CE block with known (off-line configurable) parameters based on a Matlab script prepared by other project partners in the 1st phase of the project. Then because of lack of time, that task had to be ended as such, and work was started to design Wiener filter coefficients on-line calculator for such CE algorithm. The work has been conducted by Mikko Metso / VTT (Technical Research Centre of Finland). In this chapter, both tasks are discussed in separate sub-chapters. The third sub-chapter concludes the description with a short summary.

5.1 Channel estimation

Channel estimation implementation work is to show that something reasonable simulated in project phase I [WIND210] works. For this the intention was to use a Matlab model in designing VHDL implementation for channel estimation.

The intention was to combine VHDL model definitions and the Matlab model to implement the channel estimator. Eventually this turned out to be too great of a challenge when considering the time and amount of personnel left to achieve a fully working implementation.

Below the VHDL model definitions are discussed, followed by an overview of the Matlab model in another sub-chapter.

5.1.1 Channel estimation in FPGA

Some properties for VHDL design had been defined to start with. The purpose was to do CE using 2x1D interpolation filters with pre-calculated coefficients. Target chip is a Xilinx manufactured Virtex II FPGA model XC2VP50. Initial suggestion for clock frequency was 100MHz.

Channel estimator block inputs consist of two data buses, originally from two separate antennas, both being of complex type having real and imaginary bus components with proposed width of 16 bits. Also signals signifying the validity of values on the data inputs are provided. On the output, the same input data is to be driven, delayed by the latency of the channel estimation block. Calculated CE data output is to be in-phase with the delayed input data. Again, validity signals run in parallel. In addition to this, only a test interface with necessary test data width is proposed, to enable debug runs in a logic analyser. Input buses are of the type LVDS, connected to a preceding FFT block in another FPGA circuit. Block outputs are also driven on LVDS data buses into a third FPGA that preforms LMMSE. See proposed block diagram in Figure 5-1.

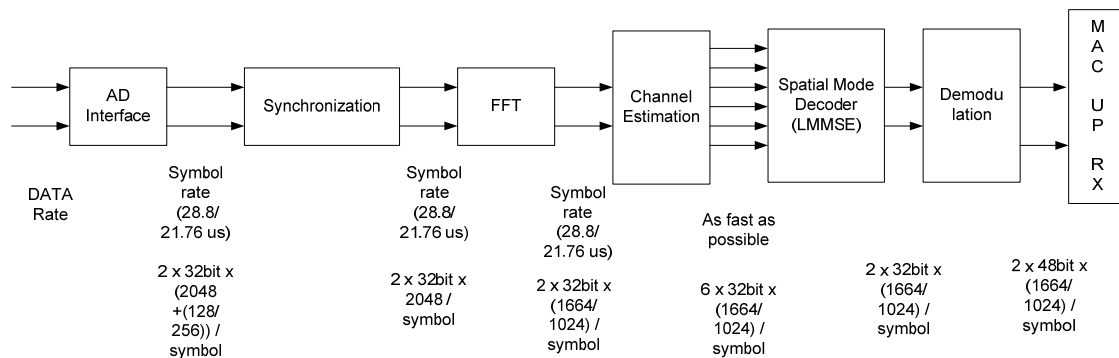


Figure 5-1 Block diagram for channel estimation implementation

Channel estimation implementation should be built for both FDD and TDD modes for a 2x2 MIMO antenna setup. The filters should be of the type matched pass-band, meaning that the filter lets all spectral components of the input pass undistorted that have delays and Doppler frequencies within this interval (=passband) and blocks all other signal components.

VHDL code is preferred parametrisable in order to easily adjust to occasional changes.

5.1.2 Channel estimation in Matlab

The Matlab model received from other project partners determines Wiener filter coefficients, generates random input data, inserts a pilot grid and runs Monte Carlo simulation to give MSE and uncoded BER plots for OFDM including channel estimation. All this is based on parameter settings where perfect knowledge of the channel statistics is assumed.

The parameters used for the model correspond to WINNER FDD mode as of project phase I [WIND210].

At the receiver channel estimation is a 2-step process:

1. Remove the modulation of pilot symbols by dividing through the transmitted pilot symbols (known at the receiver). This gives us raw channel estimates in pilot symbol positions.
2. These raw estimates are fed into an FIR interpolation filter bank. The number of filters depends on filter order and pilot spacings. The filters take a number of input values (raw channel estimates) and output one channel estimate in a desired location. In the filter bank, many small filters are used with a sliding window method; The closest set of pilots for each filter is used for the desired output. Filtering is first done in frequency direction and then in time direction.

One sub-task in transferring the Matlab model to VHDL was to pin point the parts that are related to channel estimation from all other functionalities in the model. Close study was required to outline the correct type of CE block to be implemented from the rest of the Matlab code, as well as to compare between the functionalities of the Matlab code and the CE block to be implemented in Figure 5-1.

The Matlab model uses a SISO OFDM setup which would have first had to be modified for a MIMO implementation instead. Also, the Matlab code was never intended to be used in an FPGA implementation. These were the major things that made the implementation design very cumbersome in such a short timeframe and the implementation task had to be redefined.

5.2 Wiener filter coefficient calculation on-line

There are research publications on advantages of using adaptive filtering compared to filtering with fixed-valued coefficients in OFDM channel estimation process. In one example document [SS00] an estimated 1.6dB gain in SNR can be achieved. In order to try this in practice, VHDL implementation work was started. So as the previous task was to implement a CE block that uses fixed coefficient values for the Wiener filters, the next task was to build a real-time Wiener filter coefficient calculation block without designing the rest of the CE processing. Algorithm for this on-line calculation was reviewed in the same Matlab code as in the previous task and the implementation was decided to be built based on this algorithm.

5.2.1 Selecting necessary calculations for implementation

Matlab model was first viewed to find out necessary steps to calculate Wiener filter coefficients:

1. Create a simple real-valued matrix idx_mtx according to the following parameters: amount of rows and columns in the matrix, an offset value for pilots and pilot spacing
2. Calculate a phase shift matrix for the matrix created in step 1, using another parameter f_off (frequency offset): $phase_shift = e^{i*2*\pi*f_off*idx_mtx}$
3. Calculate a sinc correlation matrix by the equation $sinc_cor_mtx = phase_shift * sinc(2*profile_range*idx_mtx)$, where $profile_range$ is a variable depending on channel properties like maximum Doppler frequency and maximum channel delay
4. Create an auto-correlation matrix according to selected parameter values by going through steps 1 to 3
5. Create a cross-correlation matrix according to selected parameter values (different than for step 4) by again going through steps 1 to 3
6. Calculate an auto-correlation matrix by adding a diagonal matrix multiplied by a value depending on noise power and pilot boost to the matrix calculated in step 4
7. Invert the matrix calculated in step 6 and multiply it with the cross-correlation matrix calculated in step 5. The multiplication result corresponds to the Wiener filter coefficient matrix.

As can be seen, rather heavy calculation is needed for the implementation to keep up with the suggested model. VHDL design was started by building basic blocks to perform the functions mentioned. These building blocks are discussed next.

5.2.2 Design of basic tools for FPGA to process needed Matlab functions

In the Matlab model there are some many basic types of calculations (additions, multiplications etc) that even for complex numbers can be quite easily implemented by using Virtex II components like 18 by 18-bit multiplier cores for multiplication or parametrisable adder cores for addition/subtraction. In addition, there are functions that require more significant design effort for implementing in VHDL. Using the basic calculations for matrices can be challenging. There are also more complex functions like raising Euler's number e to the power of a complex variable, or the inversion of a matrix mentioned in step 7 in chapter 5.2.1. Implementation work for these two functions is discussed in the following two sub-chapters.

5.2.2.1 Trigonometric functions

5.2.2.1.1 e^{ix}

To calculate functions of the type e^{ix} , where e represents Euler's number, i denotes the complex number satisfying $i^2 = -1$ and x is a complex variable, trigonometric algebra can be used. This is based on the fact that $e^{ix} = \cos(x) + i*\sin(x)$. For the implementation, two different ways to calculate sine and cosine functions were considered. One is to use lookup tables where pre-calculated values are stored based on possible input terms. An alternative way is to implement a CORDIC (Coordinate Rotation Digital Computer) algorithm that is reported [And98] to be well suited for digital signal processing and calculations.

5.2.2.1.2 CORDIC algorithm

CORDIC implementation is a pipeline structure consisting of only adder/subtractor and shift logic, and registers in between every pipeline stage. This way it can be run with high clock frequency and the amount of required logic resources is very low. For these reasons it was selected for the implementation.

In addition to reset and clock inputs, three values are fed in as calculation variables for the CORDIC VHDL component. One (Zin) denotes the angle to apply for the trigonometric sine and cosine. Outputs can also be scaled by applying scaling factors for the real (Xin) and imaginary (Yin) components in the input. Any additional logic resources are not consumed by this scaling, based on the algorithm structure. Hexadecimal value of “136EA” corresponds to a scaling value of 1 (no scaling). Output values $Xout$ and $Yout$ represent the cosine and sine values for the given angle, respectively. The pipelined architecture consists of iteration stages where the output approaches the accurate result value after each iteration. The amount of remaining angular error is always brought on to the next stage. Angular error after the last iteration stage is also driven on the output bus $Zout$ of the CORDIC component. The more pipeline stages, the longer latency for the CORDIC component.

VHDL entity port description for CORDIC top-level:

```
entity Cordic is
  port (
    RstN : in std_logic;
    Clk : in std_logic;

    --signal values are between -1 (0x20000) and 0.99999237 (0x1FFFF)
    Xin : in std_logic_vector(17 downto 0);
    Yin : in std_logic_vector(17 downto 0);
    --phase increment values theta_in are between -180 degrees (0x80000)
    --and 179.9996567 degrees (0x7FFFF)
    Zin : in std_logic_vector(19 downto 0);

    Xout : out std_logic_vector(17 downto 0);
    Yout : out std_logic_vector(17 downto 0);
    Zout : out std_logic_vector(19 downto 0)
  );
end Cordic;
```

CORDIC implementation has been simulated using a pipeline with the length of 18 stages. Modelsim simulation waveform view is shown in Figure 5-1. In this example view the sine and cosine values are counted for an angle of -120 degrees ($-2/3 \cdot \pi$ radians). In hexadecimal, being that $0x80000$ as a 20-bit number in 2's complement form represents -180 degrees, $0xAAAAB$ corresponds to $180 \cdot (-349525/524288) = -120$ degrees. As can be seen, latency between inputs and outputs is 19 clock cycles. This consists of an initial CORDIC rotation step and the 18 pipeline stages. In the implementation, a new output value is counted each clock cycle based on the inputs that were fed in 19 clock cycles earlier.

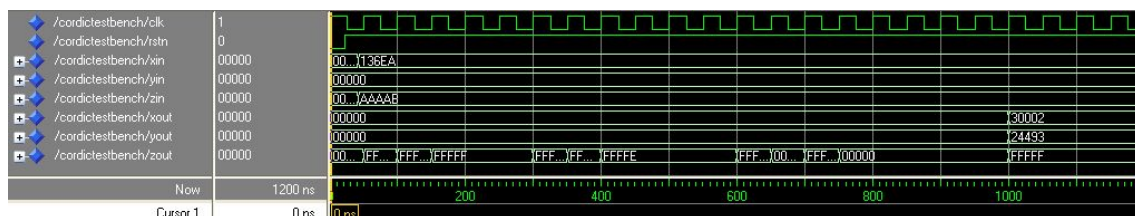


Figure 5-2 CORDIC simulation waveform view

5.2.2.1.3 Sinc

To calculate sinc functions, generally known mathematical rules can be used to open the function to the form $\text{sinc}(x)=\sin(x)/x$. This can now be implemented using CORDIC algorithm for calculating sine values and a Xilinx divider core component for the division. Another possibility is using a sinc lookup table which would save logic resources since no divider would be needed but this would in turn affect result accuracy.

5.2.2.2 Matrix inversion

Calculating matrix inversion is needed in step 7 (see chapter 5.2.1). This is a challenging work for implementation for which many suitable solutions have been searched. An algorithm called “Squared Givens Rotation” ([Döh91], [KCD05] and [EL06]) has been a hot topic among telecom researchers with promising results. This is an algorithm that requires the use of floating point arithmetic but is fast and economical in resourcing point of view. The basic idea is to build a QR-decomposition for the input data matrix, consisting of an orthogonal matrix Q and an upper triangular matrix R. Then the triangular matrix R is inverted using a recurrence algorithm and then multiplied by the matrix Q to form the inverted input matrix.

To start with the implementation of a matrix inverter component, Xilinx floating point core component was simulated to be used in Squared Givens Rotation calculations. Then the literature was further studied for a deeper knowledge of the algorithm. At this point, a parametrisable block in C code was proposed for the task by other project partners and so there was no more need for matrix inversion block. The current task was redefined to include Wiener filter on-line coefficient calculation implementation without matrix inversion.

5.2.3 Selecting parameters for implementation

Necessary parameters for creating the coefficient calculation block were searched for in the Matlab model. The intention was to have the VHDL code parametrisable in order to provide a flexible design. For this, upper and lower limits for each parameter also needed to be known. Parameter listings gathered from the Matlab model are presented below, with the default value for each parameter in parentheses:

- number of Wiener filter coefficients in frequency direction (default 16)
- pilot spacing in frequency direction (default 4)
- offset of first pilot in frequency and time directions, with respect to subcarrier/OFDM symbol (default 0,0)
- maximum expected channel delay (default 66)
- oversampling factor (default 10)
- length of the guard interval (default 640)
- FFT window size (default 5120)
- number of subcarriers (default 416)
- synchronization offset in time (default -10)
- mean noise power (default 0.1)
- mean power of a M-QAM modulated symbol (default 1)
- energy per symbol (linear scale) (default 10)
- energy per bit (linear scale) (default 10)
- pilot boost (default 1)

Also properties for other values, such as matrix cell data width, needed to be predefined. Due to small amount of time left and the wide gap between the Matlab model and the awaited VHDL component, these values have not been found.

5.3 Conclusions

Like the first task of implementing a channel estimator, also the Wiener filter coefficient on-line calculator based on a Matlab model was a too heavy task to implement in this time scale. A model that would have been optimised for implementation, where certain amount of approximations could have been applied, would be needed in addition to more thorough definition for the low-level VHDL component parameters.

6 Synchronization

6.1 Synchronisation symbol

The timing synchronisation symbol is built in the frequency domain by modulating one out of four subcarriers. This subcarrier mapping brings a 4 times repetition pattern in the time domain as shown in Figure 6-1. The result of autocorrelation formula $A(n)$ given in Eq 6.1 remains constant during $2N_{pre} + N_{CP}$ samples (where N_{pre} denotes the repetition length and N_{CP} the cyclic prefix length). This time window is called flat region. Due to the cyclic structure, the flat region also occurs in case of multipath channel.

$$A(n) = \sum_{i=0}^{N_{pre}-1} r^*(n-i)r(n-i+N_{pre}) \tag{6.1}$$

where $r(n)$ is the received sample at the timing index n .

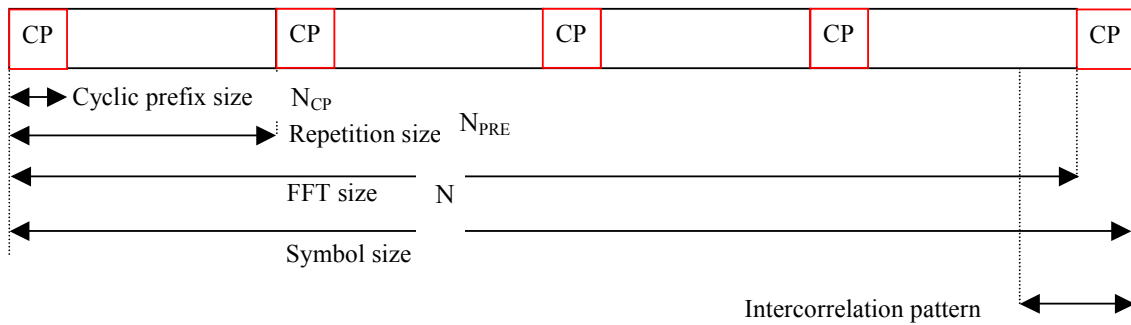


Figure 6-1: time domain synchronisation symbol

The indices of the 1024 non-zero subcarriers (out of 2048 for FDD and TDD modes) are defined by:

$$\begin{cases} k \% 4 = 0 \\ k \in [1; 512] \\ k \in [1536; 2047] \end{cases}$$

where % is the modulo operator.

The baseband spectrum is the following:

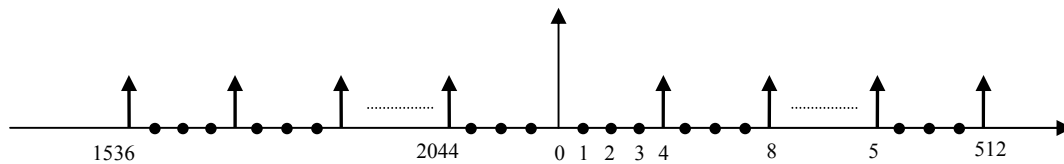


Figure 6-1: Baseband synchronisation symbol spectrum

6.2 Synchronisation algorithm

The synchronization aims at finding a time reference for the FFT vector for OFDM demodulation and at estimating the pre-FFT CFO. It is processed on the fly and runs continuously. The algorithm is updated every received sample. It is based on an autocorrelation processing, whose role is to determine a window in which the cross-correlation between the input signal and the known sequence (defined in Figure 6-1) is analysed. Figure 6-2 shows the synchronisation process (the signals and parameters used in the algorithm are in italic): Definitions of terms used in Figure 6-2 and in the algorithms are given below:

- *Autocorrelation* is the autocorrelation. The window for cross-correlation analysis is opened when the result of the autocorrelation exceeds the pre-defined threshold (*THRESHOLD_AUTO*), and is closed when it goes under the same threshold. In Figure 6-2, the window is delimited by the vertical lines.
- *Fsm* is the state machine that manages the algorithm. When the autocorrelation result exceeds the threshold, a sample counter runs till the end of the window (the *fsm* is in *count* state). The flat region is found when the counter reaches the programmed value (*N_HOLD*).
- *Intercorrelation* is the cross-correlation. When the input data flow is in phase with the sequence, peaks appear. If these peaks exceed the pre-defined threshold (*THRESHOLD_INTER*) and are within the window, there are taken into account (see next bullet).
- *Peakpulsesw* indicates when a valid peak is found.
- *Framestart* is the pulse that indicates the first sample of the OFDM symbol. When the window gets closed, and 3 cross-correlation peaks are found, the state machine *fsm* goes to *synch* state which means that the system is synchronised. The end of the synchronisation symbol is precisely given by the position of the last cross-correlation peak. The *framestart* pulse is the former position, **delayed** by *DataDelay*, a user defined number of samples, to take into account the processing time.

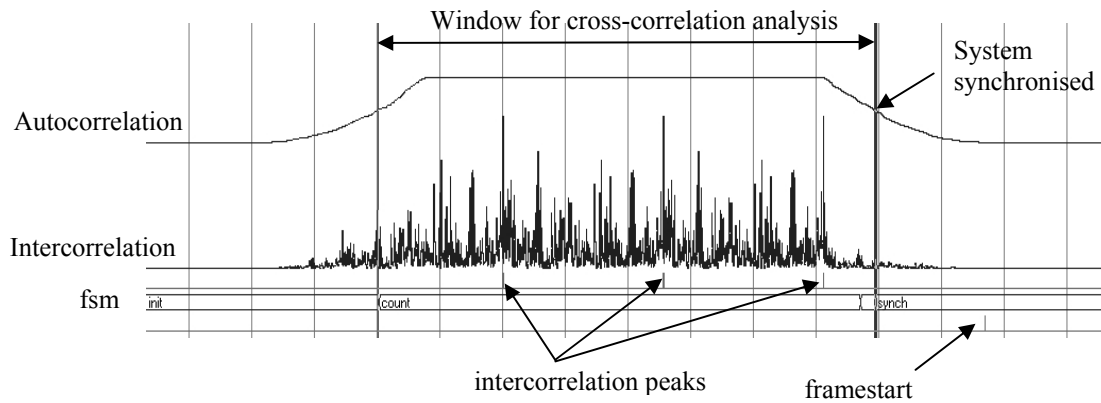


Figure 6-2: Ideal noiseless case of autocorrelation and crosscorrelation profile

The autocorrelation also enables to get the RF oscillator frequency shift. Actually, during the flat region

detection, we have $\arg(A(n)) = 2\pi \frac{N_{pre}}{N} \frac{\Delta f}{\Delta f_{sc}} = \varphi_0$ whatever n may be.

where

$N_{pre} = 512$ is the repetition length,

$N = 2048$ is the FFT length

$\frac{\Delta f}{\Delta f_{sc}} = CFO$ is the Carrier frequency offset defined as the ratio between Δf the TX and RX oscillator

shift, and Δf_{sc} the subcarrier spacing.

The correction has to be applied to every sample. The rotation applied to sample #n is

$2\pi \cdot n \cdot \Delta f T_s = 2\pi \cdot n \cdot \frac{\Delta f}{N \Delta f_{sc}}$ where T_s is the sampling period. It comes that the correction value is

$n \frac{\varphi_0}{N_{pre}}$.

The computation of the pre-CFO correction requires the following operators:

- Cartesian to polar conversion,
- Phase division by N_{pre} ,

- Accumulator for sample phase correction
- Polar to cartesian conversion to carried out samples correction.

6.3 Synchronisation module

The top level entity of the synchro module is the following:

ENTITY WINNER_synchro_td IS

```

PORT (
  clk                : IN std_logic;
  reset              : IN std_logic;
  SetGain            : IN std_logic;
  Synch_ok           : OUT std_logic;

  -- prog
  fdd_tdd            : IN std_logic;
  THRESHOLD_AUTO    : IN std_logic_vector(7 DOWNTO 0);
  THRESHOLD_INTER    : IN std_logic_vector(7 DOWNTO 0);
  N_HOLD            : IN std_logic_vector(11 DOWNTO 0);
  N_SYMBOL_SYNC     : IN std_logic_vector(7 DOWNTO 0);

  CFOset             : IN std_logic;
  CoarseCFOTest     : IN std_logic_vector(15 DOWNTO 0);

  DataDelay          : IN std_logic_vector(15 DOWNTO 0);
  --
  EnSample           : IN std_logic;
  Iin                : IN std_logic_vector(11 DOWNTO 0);
  Qin                : IN std_logic_vector(11 DOWNTO 0);

  FrameStart        : OUT std_logic;
  SymbolStart       : OUT std_logic;
  Iout               : OUT std_logic_vector(11 DOWNTO 0);
  Qout               : OUT std_logic_vector(11 DOWNTO 0);
);
END WINNER_synchro_td;
```

Description of the signal:

<i>Name</i>	<i>Description</i>
Clk	Master clock
Reset	Synchronous reset
SetGain	For Gain Control module connection. When equal to '1' the synchronisation module is reset. This signal is needed while the system sets the RF gain. When equal to '0' the synchronisation module can run.
Synch_ok	For Gain Control module connection. Indicates when the synchronisation module is in the synchronized state. This state duration is determined by the parameter N_SYMBOL_SYNC : during this time, the system does not try to refresh synchronization.
EnSample	Validate the input data. Generally has to be set to '1'.
Iin	Real part of the input data

Qin	Imaginary part of the input data
FrameStart	Pulse that indicates the first sample of the frame
SymbolStart	Pulse that indicates the first sample of the OFDM symbol
Iout	Real part of the synchronised data flow
Qout	Imaginary part of the synchronised data flow

Description of the user parameters:

<i>Name</i>	<i>description</i>	<i>Default value to set</i>
fdd_tdd	'0' : FDD mode '1' : TDD mode	
THRESHOLD_AUTO	Autocorrelation threshold	HEX"80" for 50% threshold
THRESHOLD_INTER	crosscorrelation threshold	HEX"80" for 50% threshold
N_HOLD	Number of autocorrelation samples that have to exceed the autocorrelation threshold to determine the flat region.	HEX"400"
CFOset	If '1' the CFO estimator is set with CoarseCFOTest. If '0' the module estimates the pre-FFT CFO. If no pre-FFT correction is needed, put CFOset='1' and CoarseCFOTest=HEX"0000"	'0'
CoarseCFOTest	Programmable CFO value when CFOset='1', CFO = CoarseCFOTest/16384	HEX"0000"
DataDelay	Programmable delay for data resynchronisation. Default value enables to get the synchronisation pulse in the middle of the guard time	HEX"139F"(=5023) for FDD mode. HEX"13DF"(=5087) for TDD mode;

7 Trials validation

In this chapter, we describe requirements of trials, system model build and simulation results we get. We will describe basics concepts e.g. modulation, forward error correction coding, spatial processing, etc, in following sections.

Grid of fixed beam is selected as one of the transmission scheme used in baseline of WINNER II. In this section we first introduce a fixed beamforming (BF) technique for orthogonal frequency division multiplexing (OFDM) systems. The basic idea of beamforming is that the phase of the transmitted signal is shifted according to the user direction, while at the receiver side, the phase shifting performs again to maximize the received signal-to-noise ratio (SNR). We also propose a discrete Fourier transform (DFT) based estimation algorithm which can estimate the beam domain channel. To reduce the computational complexities and feedback signalling, we do beamforming and channel estimation based on the chunk, which is a unit that occupies several symbol time intervals and subcarriers. The simulation results show the performance of the proposed beamforming technique associated with channel estimation algorithm. We also compare the beamforming technique we proposed with Alamouti space-time block coding (STBC).

A simple fixed beamforming technique based on Butler matrix has been introduced and studied in [Yli06]. It utilizes multiple antennas at the transmitter and receiver sides. By knowing the direction of the users, the transmitted symbols are multiplied by a weighting vector, then mapped into parallel spatial streams which have the same symbols but a different phase shift value. By shifting the phase of the symbols at the transmitter, a directional beam is thus formed to the direction of the user. Compared to the traditional sectoral transmission, the beamforming has a narrower beam with stronger power at the receiver, i.e., the power is more concentrated on the user, which can increase the signal to noise ratio and reduce the interference to other users.

7.1.1 Deployment scenario

Basic deployment scenarios are described below:

- 1 base station (BS), wide area (FDD)
- cellular, hexagonal layout
- base station above rooftop, 25m
- 2 or 4 antennas per sector
- antennas configuration: linear array
- antenna element spacing: 0.5λ
- 1 user terminal (UT)
- Up to 30 kmph user mobility within wide area
- 2x2 spatial scenario in DL and UL respectively
- DL: OFDM(A), UL: OFDM(A)
- Web browsing or file download

7.1.2 Transceiver specifications

7.1.2.1 Forward error correction and modulation

Convolutional codes are used as baseline encoder/decoder.

Code rate: $\frac{1}{2}$,

Viterbi decoder

BPSK, QPSK, 16-QAM, 64-QAM are used.

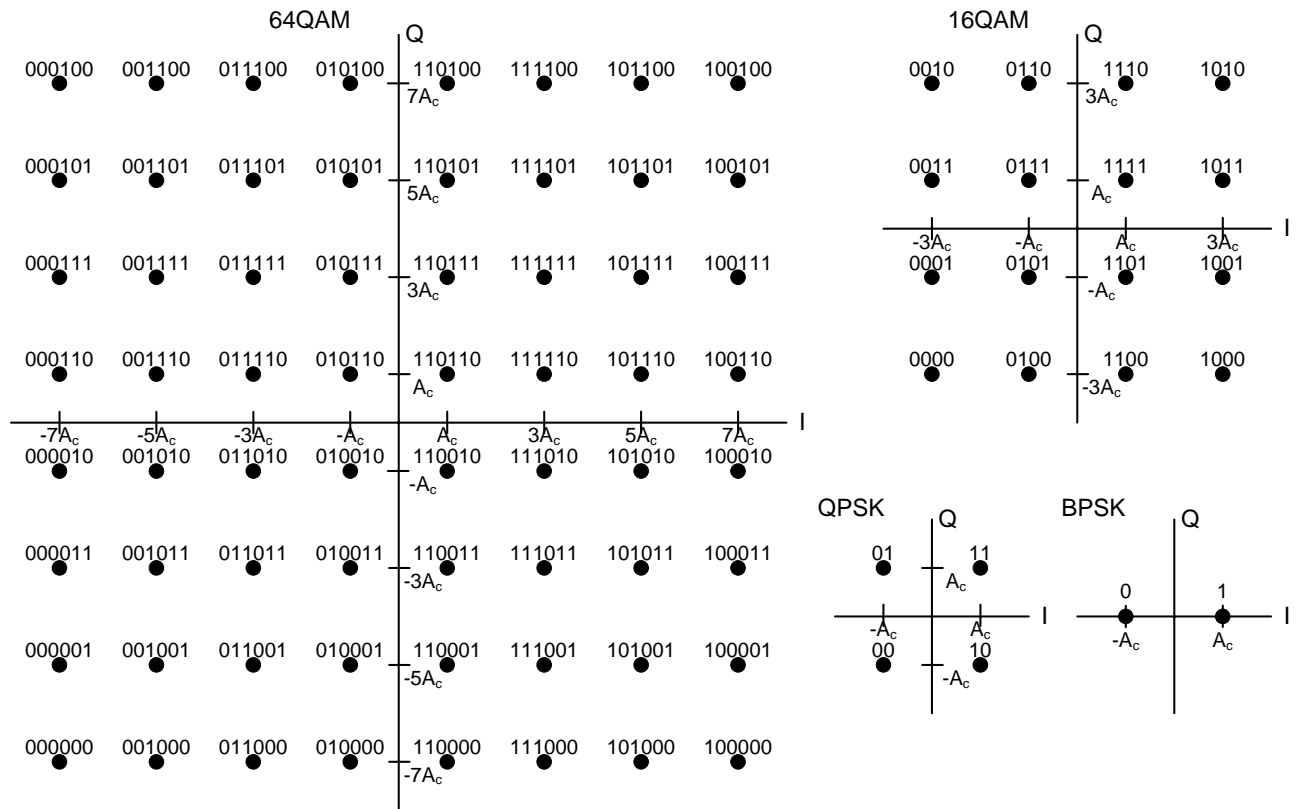


Figure 7.1 Gray mapping & constellations

7.1.2.2 Multi-carrier technique

Block signal processing in the frequency domain using (inverse) fast Fourier transforms (FFT/IFFT) at transmitter and receiver is the choice for WINNER II baseline simulation. The table below gives the initial OFDM parameters.

Table 7.1 OFDM parameters

	Base Coverage Urban
Subcarrier distance Δf	39062.5 Hz
Useful symbol duration T_N	25.6 μ s
Guard interval T_G	3.2 μ s
Total symbol duration	28.8 μ s
Number of subcarriers	1024

7.1.2.3 Space-time processing

7.1.2.3.1 Space-time block coding

Space-time block coding (Alamouti scheme) is used in the baseline system. The Alamouti STBC can be summarized as follows. The scheme considered here uses two transmit and received antennas. At a given symbol period, two symbols s_1 and s_2 (each with energy $E_s/2$) are simultaneously transmitted from the antennas 1 and 2, respectively. During the next symbol period signal $-s_2^*$ is transmitted from antenna 1 and signal s_1^* is transmitted from antenna 2, each again with symbol energy $E_s/2$. The output of the encoder is

$$\begin{aligned} c_1(t) &= s_1 & c_1(t+T) &= -s_2^* \\ c_2(t) &= s_2 & c_2(t+T) &= s_1^* \end{aligned}$$

where $c_i(t)$ is the corresponding output symbol to antenna $i = \{1, 2\}$ and T is the symbol period.

Assuming that channel is constant across two consecutive symbols, the signal received by antennas 1 and 2 can be expressed as follows

$$\begin{aligned} y_1 &= h_1 s_1 + h_2 s_2 + n_1 \\ y_2 &= -h_1 s_2^* + h_2 s_1^* + n_2 \\ y_3 &= h_3 s_1 + h_4 s_2 + n_3 \\ y_4 &= -h_3 s_2^* + h_4 s_1^* + n_4 \end{aligned}$$

where the h_i ($i=1,2,3,4$) is the channel between the transmit and receive antennas. At the receiver decoding is done based on the maximal ratio combining (MRC) algorithm.

7.1.2.3.2 Grid of fixed beams

Beamforming techniques use multiple antennas to focus beams in certain directions to increase the SNR and reduce the interference. Fixed beam MIMO transmission will be one of the baseline WINNER concepts.

For transmission to a certain user the beamforming vector \mathbf{v} is chosen from a predefined static set of complex numbered antenna weights, so-called beams. The receive vector of each user and subcarrier has the following form,

$$\underbrace{\mathbf{y}}_{M_R \times 1} = \underbrace{\mathbf{H}}_{M_R \times M_T} \underbrace{\mathbf{v}}_{M_T \times 1} \underbrace{p}_{1 \times 1} \underbrace{s}_{1 \times 1} + \underbrace{\mathbf{z}}_{M_R \times 1}$$

where $\mathbf{V} \in \{\mathbf{v}_1 \quad \mathbf{v}_2 \quad \dots \quad \mathbf{v}_I\}$, and I denotes the number of available beams.

\mathbf{H} is the MIMO channel matrix, p the transmit power, s the transmitted symbol and \mathbf{z} the vector of interference plus noise.

The antenna weights \mathbf{v}_i (with non tapered beams) are calculated from the main beam direction \mathcal{G}_i of beam i and from the m -th transmit element position d_m for all M elements, with $k = 2\pi/\lambda$ being the wave number, according to:

$$\mathbf{v}_i(\mathcal{G}_i) = \frac{1}{\sqrt{M}} [v_{i1}(\mathcal{G}_i) \quad v_{i2}(\mathcal{G}_i) \quad \dots \quad v_{iM}(\mathcal{G}_i)]^T \quad \text{with}$$

$$v_{im}(\mathcal{G}) = \exp(-jkd_m \sin(\mathcal{G}_i)).$$

The uniform linear array with $\lambda/2$ spacing has the element positions $d_m = \pm \frac{1}{4}\lambda, \pm \frac{3}{4}\lambda, \left(\pm \frac{5}{4}\lambda, \pm \frac{7}{4}\lambda\right)$.

\mathbf{v}_i can be chosen from butler matrix, which makes beams orthogonal in propagation direction. For 2 beams, the butler matrix is $[1 \ j; 1 \ -j]$, and for 4 beams, the butler matrix is

$$\begin{bmatrix} 1 & -1/\sqrt{2} - j/\sqrt{2} & +j & +1/\sqrt{2} - j/\sqrt{2} \\ 1 & +1/\sqrt{2} - j/\sqrt{2} & -j & -1/\sqrt{2} - j/\sqrt{2} \\ 1 & +1/\sqrt{2} + j/\sqrt{2} & +j & -1/\sqrt{2} + j/\sqrt{2} \\ 1 & -1/\sqrt{2} + j/\sqrt{2} & -j & +1/\sqrt{2} + j/\sqrt{2} \end{bmatrix}$$

7.1.2.4 Chunk wise processing

The basic space-time-frequency unit in WINNER II is chunk, which is similar with the concept of ‘physical resource block’ in 3GPP LTE. In FDD mode, a chunk comprises 8 subcarriers by 12 OFDM symbols or 312.5 kHz x 345.6 μ s. In the TDD mode, the chunk dimension is 16 subcarriers by 5 OFDM symbols or 781.25 kHz x 108.0 μ s.

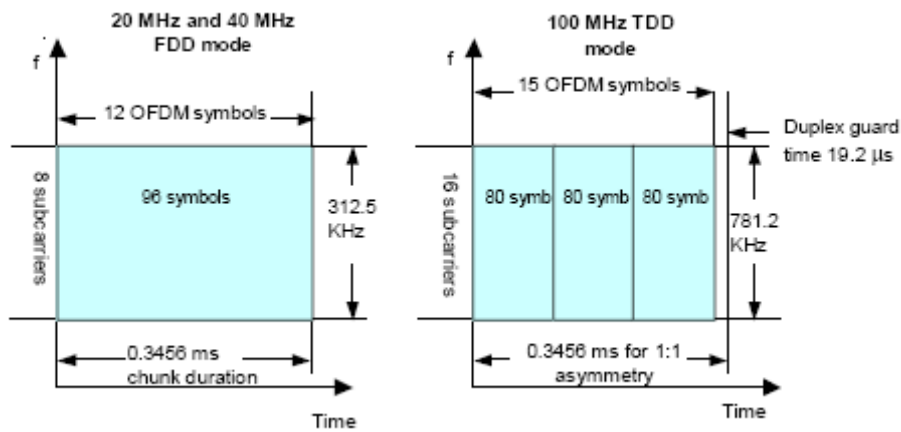


Figure 7-1.2 Chunk sizes in physical layer modes

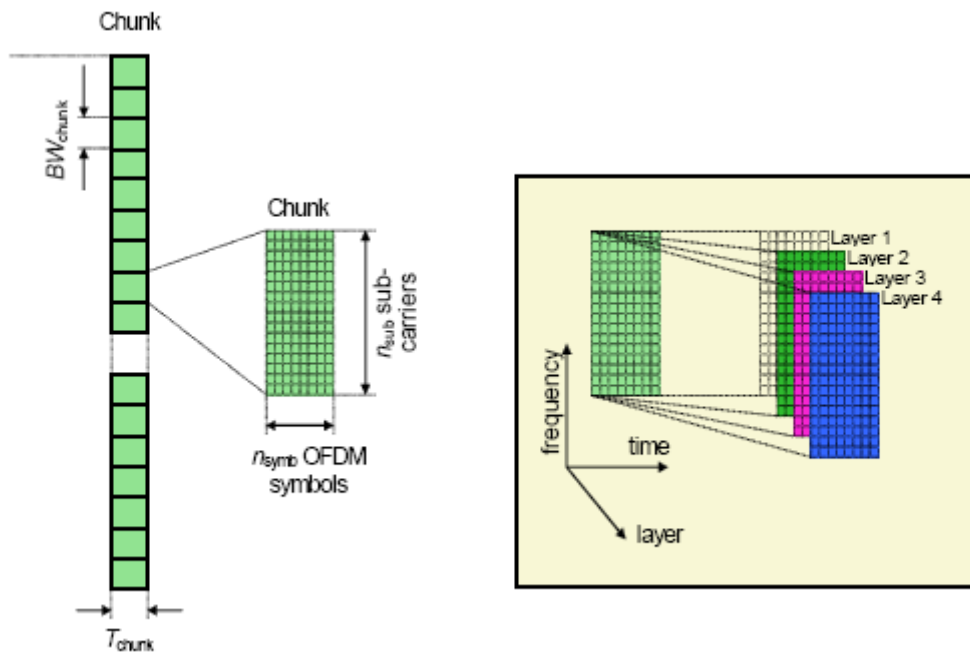


Figure 7.3 Downlink physical channel structure and chunks

7.1.2.5 Channel estimation at the receiver

Pilot aided channel estimation is used in WINNER II baseline simulation. The scattered pilots are separated 7 subcarriers away from each other in frequency direction and 11 symbols away in time direction, i.e., in one chunk, there are 4 pilots.

When pilot is being transmitted from one antenna, the other antenna transmit zero to avoid interference in channel estimation at the receiver.

DFT based channel estimation is used. In time direction, linear interpolation is used to estimate the channel between two symbols that carrying pilots.

7.1.3 Channel model

The channel model is WINNER phase I model (WIM), and the propagation scenario used is C2, typical urban macro-cell.

7.1.4 Expected performance

The expected results for baseline trials are:

- BER vs. SNR performance in WIM for different modulation scheme and user speed.
- BER vs. SNR performance with IFFT-based channel estimation.
- BER vs. SNR performance with delayed feedback.

7.2 System model

Fig. 7.3 shows the overview of our proposed 2 by 2 MIMO-OFDM systems. In OFDM systems, the data is first encoded and modulated. The modulated symbols are then OFDM modulated by inverse discrete Fourier transformed (IDFT), and cyclic prefix (CP) is added to each OFDM symbol to prevent inter symbol interference (ISI). It is assumed that the length of the CP is larger than the delay spread of the multipaths. The OFDM symbol stream is multiplied by a beamforming vector to form a directional beam. In the receiver, maximum ratio combining is performed, which means that the received symbols are multiplied by an optimal combining vector to match the antenna weights to the corresponding radio channel responses. Then CP is removed and symbols are decoded/demodulated. We assume that the OFDM symbols are perfectly synchronized and that the subcarriers are orthogonal to each other, i.e., no inter carrier interference (ICI) is present.

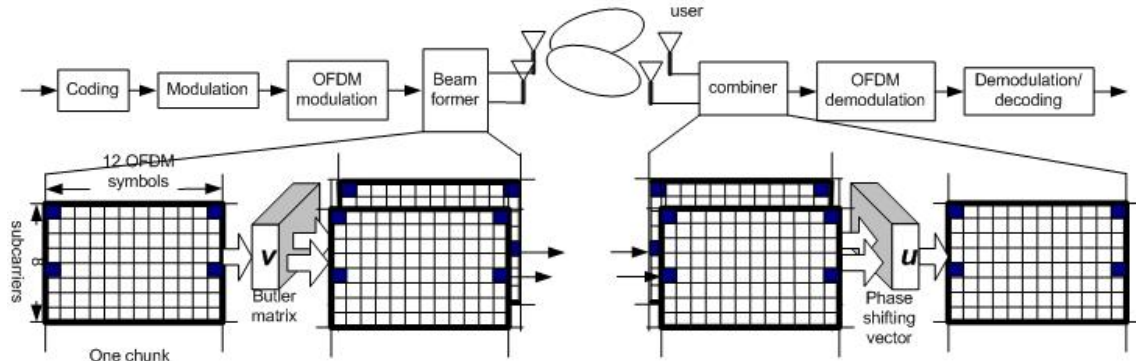


Figure 7.3 Block diagram of fixed beamforming.

The basic idea of fixed beamforming is by multiplying a beam forming vector \mathbf{v} , a data symbol is mapped into several phase shifted symbol vectors,

$$\mathbf{r}_l^k = \mathbf{H}_l^k \mathbf{v}_l s_l^k + \mathbf{n}_l^k,$$

where \mathbf{r}_l^k , \mathbf{H}_l^k , \mathbf{v}_l , s_l^k and \mathbf{n}_l^k are received signal vector, channel frequency transfer function, the beam forming vector, the transmitted symbol and the noise term on the k th subchannels during the transition time n in chunk index l respectively. It is equivalent to consider the system in the beam domain.

$$\mathbf{r}_l^k = \mathbf{b}_l^k s_l^k + \mathbf{n}_l^k,$$

where

$$\mathbf{b}_l^k = \mathbf{H}_l^k \mathbf{v}_l + \mathbf{n}_l^k,$$

can be considered as the beam domain channel. To form two fixed beams for two transmitting antennas, the beam forming vector \mathbf{v}_l , can be chosen from the Butler matrix

$$\mathbf{V} = [\mathbf{v}_1 \quad \mathbf{v}_2] = \begin{bmatrix} 1 & 1 \\ j & -j \end{bmatrix}.$$

Note that the norm of the beam forming vector is 2 corresponding to the beamforming gain. The imaginary unit, j , stands for 90 degrees phase shift of the signal. The criterion to choose \mathbf{V}_l is made based on the direction of the user. The user estimates the channel and feeds back the decision to the BS to form the best beam. For an optimal beam, along with the maximum ratio combining (MRC) receiver, the signal to noise ratio (SNR) at user side will be maximized.

The MRC is a amplitude and phase shifting procedure of the received signal. The received signal is multiplied by a phase shifting vector \mathbf{u}_i . Since the receiver has no knowledge of which beam forming vector is used at Tx, so two receiving phase shifting vectors are calculated,

$$\mathbf{u}_i = \frac{\mathbf{b}_i}{|\mathbf{b}_i|} = \frac{\mathbf{H}\mathbf{v}_i}{|\mathbf{H}\mathbf{v}_i|}, i = 1,2.$$

The one that maximizes the SNR is chosen as receiving phase shifting vector,

$$\mathbf{u}_l = \arg \max_{\mathbf{u}_i \in \{\mathbf{u}_1, \mathbf{u}_2\}} \left| \frac{\mathbf{u}_i^H \mathbf{b}_i}{|\mathbf{b}_i|} \right| \in \mathbb{C}^{N_R \times 1}.$$

The algorithm is illustrated in the Fig. 7.4. The estimated received signal with maximum ratio combiner can be expressed as

$$\hat{s}_{ln}^k = \mathbf{u}_l^H \mathbf{r}_{ln}^k = \mathbf{u}_l^H \mathbf{b}_{ln}^k s_{ln}^k + \mathbf{u}_l^H \mathbf{n}_{ln}^k.$$

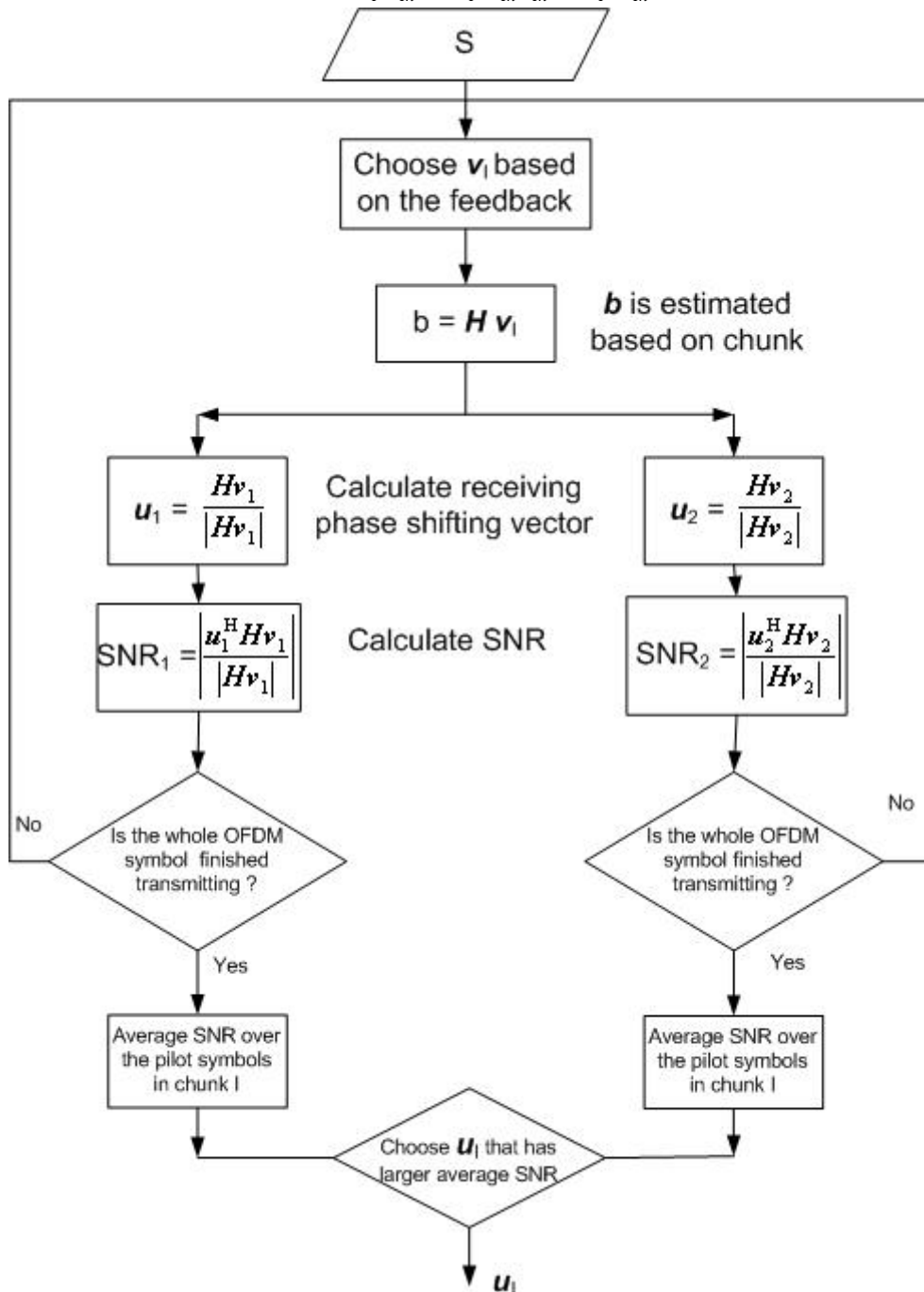


Figure 7.4 algorithms to calculate received phase shifting vector.

7.3 Beam domain channel estimation

In this section, we propose a simple beam domain channel estimation method based on discrete Fourier transform(DFT) [Aue03] . The pilots are distributed in the chunk, as shown in Fig. 7.3, where the dark squares are the pilot grid. The beam domain subchannel at a pilot position can be estimated as

$$\hat{\mathbf{b}}_{l_n}^k = \mathbf{r}_{l_n}^k / \hat{s}_{l_n}^k,$$

where $/$ denotes element-wise division and $\hat{s}_{l_n}^k$ denotes known transmitted pilot symbol. The benefit we get from beam domain channel estimation is that by multiplying a 2 by 1 beam forming vector \mathbf{v}_l , the 2 by 2 channel matrix $\mathbf{H}_{l_n}^k$ is transformed into a 2 \times 1 vector $\mathbf{b}_{l_n}^k$. Thus, there is no need for matrix inverse computation as in normal channel estimation technique which saves the cost of the receiver. The other advantage is that it does not require the knowledge of channel statistics.

Once we know the beam domain channel transfer functions at pilot positions, the transfer function for all subcarriers can be obtained by interpolation in frequency domain, i.e., DFT-based algorithm. First, $\tilde{\mathbf{B}}_{l_n}$ which composed of N_p estimated vectors $\mathbf{b}_{l_n}^k$, is N_p -point inverse Fourier transformed, where N_p is the number of pilot subcarriers in one OFDM symbol .

$$\mathbf{g}_{l_n} = \text{IDFT}\{\tilde{\mathbf{B}}_{l_n}\} \in \mathbb{C}^{N_p \times N_R}$$

Then $(N_c - N_p) \times N_R$ zeros are padding to \mathbf{g}_{l_n} . After that, N_c -point DFT transforms \mathbf{g}_{l_n} into beam domain channels for all subcarriers,

$$\hat{\mathbf{B}}_{l_n} = \text{DFT}\left\{ \begin{bmatrix} \hat{\mathbf{g}}_{l_n} \\ \mathbf{0}_{(N_c - N_p) \times N_R} \end{bmatrix} \right\} \in \mathbb{C}^{N_c \times N_R}$$

Since we get the all beam domain subchannels for OFDM symbols carrying pilots. In the next stage, we perform linear interpolations between two adjacent pilots carrying channels that we estimated. The time direction linear interpolation is given below,

$$\hat{\mathbf{B}}_{l_{n_i}} = \hat{\mathbf{B}}_{l_{n_1}} + \frac{n_i - n_1}{\Delta t} \left[\hat{\mathbf{B}}_{l_{n_1}} - \hat{\mathbf{B}}_{l_{n_2}} \right], n_1 \leq n_i \leq n_2,$$

where n_1 and n_2 are time indices of OFDM symbols that carrying pilots within chunk l , and Δt is time gap between two pilots-carrying symbols. In our case, $n_1 = 1$, $n_2 = 12$ and Δt is 12. The frequency direction DFT interpolation and time direction interpolation is illustrated in Fig. 7.5.

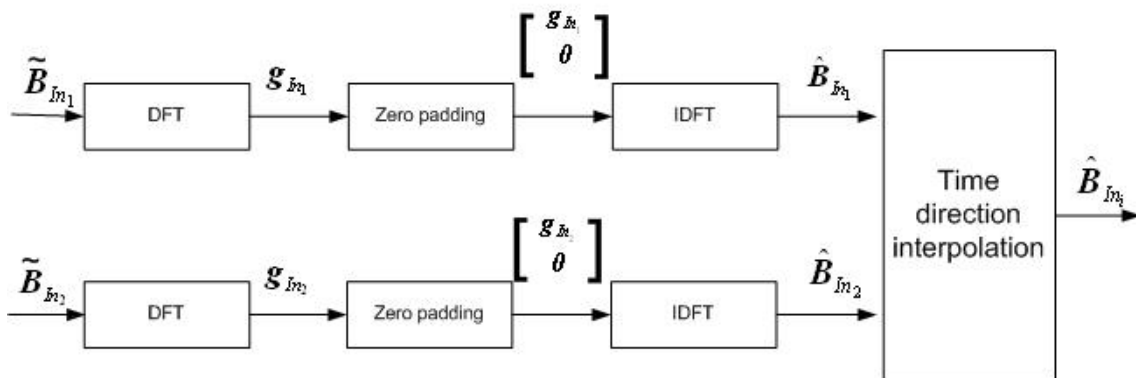


Figure 7.5 block diagram of channel estimation for fixed beamforming.

7.4 Simulation results

The simulation parameters are based on the baseline concept of WINNER phase II. The channel model used in simulation is WINER phase I channel model [WIM]. The main parameters are listed in the Table 7.2. The effect of feedback error and delay is not considered here.

Table 7.2 Simulation parameters.

Number of antennas	2 × 2
Channel coding	Convolutional
Modulation	QPSK
System bandwidth	2 × 50 MHz
Subcarrier distance	39062.5 Hz
Number of OFDM subcarriers	1024
Chunk size	12 symbols × 8 subcarriers
Channel model	WINNER phase I model (WIM)
Scenarios	Base coverage urban, 'C2'
Duplexing mode	FDD
User speed	3km/h, 120 km/h

We present the BER performance of 2 × 2 system with 2 and 3 fixed beams at different mobile speed. We find that system with 3 fixed beams provide similar performance than the system with 2 fixed beams. Also, the BER performance degradation due to the error of channel estimation can be found in the results. From Fig. 7.6 we find that for 2 fixed beams, for user speed at 3 km/h and 120 km/h, the BER performance is almost the same when channel state information (CSI) is perfectly known at the receiver. However, with channel estimation, due to the effect of Doppler shift, the performance of user speed at 3 km/h is about 5 dB better than the user speed at 120 km/h when SNR is 10⁻². Fig. 7.7 shows the results for 3 fixed beams. With channel estimation, the BER difference between 3km/h and 120km/h is 2.2 dB. If we compare the 2 and 3 fixed beams with perfect CSI at speed of 3 km/h and BER of 10⁻², we find that 3 fixed beams system dose not provide much better performance than 2 fixed beams system does. Alamouti STBC with 2 × 2 antennas set up is also simulated as shown in Fig. 7.7, and all other parameters are the same with BF. We do not compare the channel estimation performance of STBC since the pilot structure and estimation algorithms are different in STBC and BF. The results show that BF is better than STBC, and the performance gain is about 3 dB which comes from the antenna array gain.

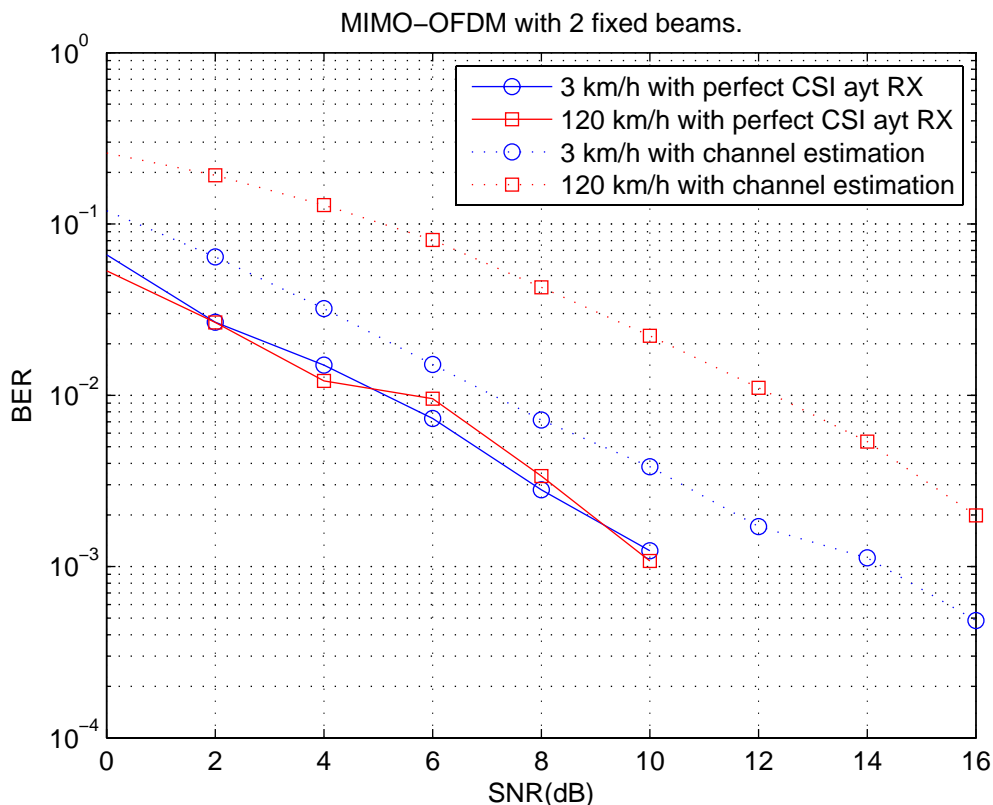


Figure 7.6 BER of 2 fixed beams.

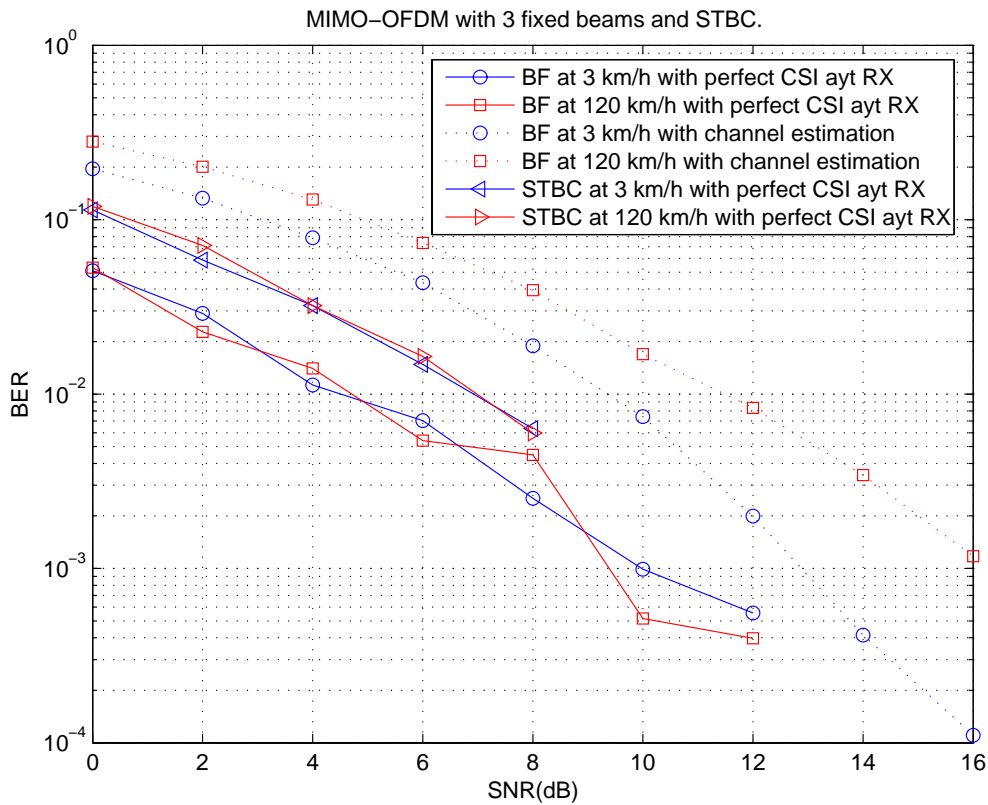


Figure 7.7 BER of 3 fixed beams and STBC.

In Fig 7.8, affect of channel estimation error is shown. In Fig. 7.9, we present comparison of the BER performance of a 2×2 QPSK modulated system at a user speed of 3 km/h, obtained under the assumptions of feedback without delay, delayed feedback and perfect CSI without delay respectively. For the situation with channel estimation and without delay, the system performs about 4dB worse than for the perfect CSI, which is due to the degradation caused by channel estimation error. The performance of delayed feedback is worse than that of the feedback without delay. As the SNR is increasing, the gap between these two are getting larger.

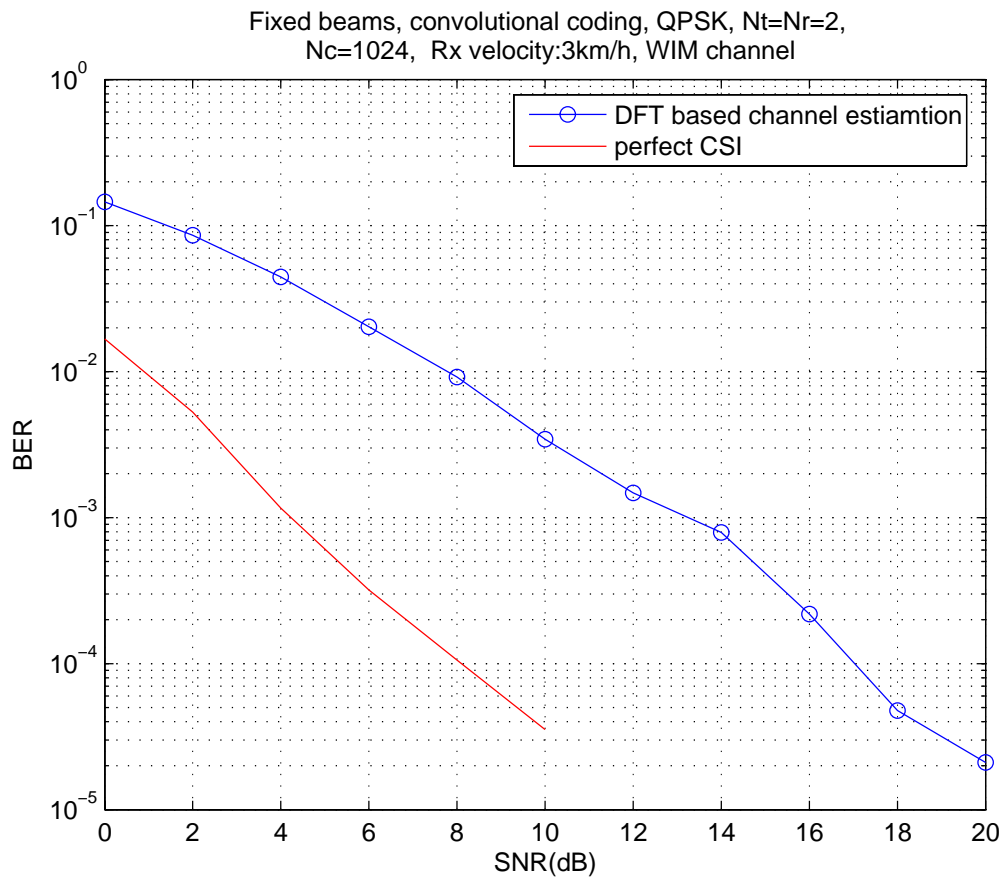


Figure 7-8 Comparison of perfect CQI at Rx and estimated channel.

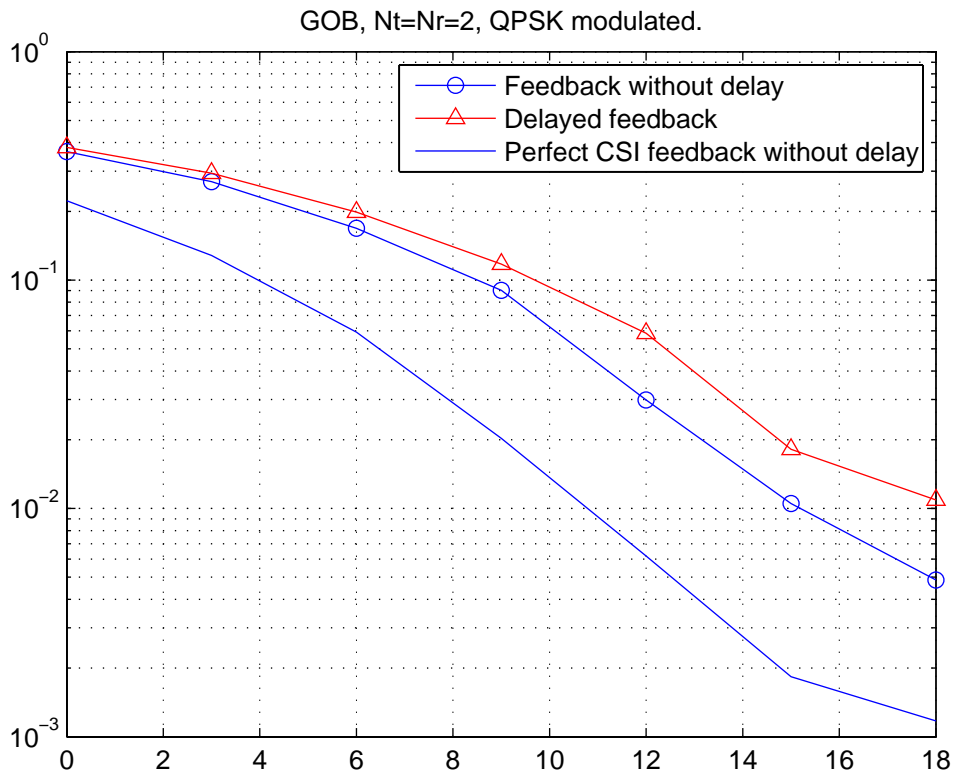


Figure 7.9 Performance comparison of CSI with and without delay.

8 Conclusion

8.1 Trial targets and evaluation

General objectives of T12 from Annex 1 of project agreement were elaborated in previous report of PHY trials [WIN2IR6121] into the following list, with which the current trials status and achieved objectives can then be contrasted. To be precise, although phase II of WINNER is ending, it is expected that work on the system and code integration continues in some way.

8.1.1 Objective 1

The description of this objective was “Preparation of a plan for the full WINNER system trials to be performed in Phase III”. It included the following subitems:

- Phase III targets: The experience and direct products in Phase II trial can be used directly or indirectly for planning Phase III full trials
- trial implementation
- validation concept

The realization of these goals could be here evaluated as follows: The phase III target is a vague one, but indeed many parts of the existing code base could be used for the possible continuation of the project, e.g. the LDPC codec and code related to adaptivity. The code base has been built with parametrizability in mind, so it will be able to support changes in the MAC/PHY, to a degree.

The trials implementation for phase III can be found in the [WIN2D6121]. The validation concept developed in the PHY trials would allow validation for certain aspects of link adaptivity, with approaches explained in chapter 2. Unfortunately, during the time of the submission of this paper measurement results are not available, as some integration is still left, and events requiring the attendance of the trials take some time out from the lab.

8.1.2 Objective 2

The description of this objective was “Implementation and validation of selected functionality of WINNER system concept”. The following subitems were included:

- Validation targets: There shall be a validation concept which allows for validation of WINNER concepts
- WINNER II channel models are used thoroughly in testing
- Synchronization and channel estimation imperfections, integer mathematics are modeled
- An integration of WINNER phase I/II algorithms allows for detailed study of link performance

The first subitem on this list is largely the same as the last subitem in chapter 8.1.1. Again one has to remember that full review on a WINNER link is not possible, as many needed aspects and features planned for a reference design are not available. The second subitem was successfully achieved, although the nature of the trials implementation does not allow high mobile speeds. The third item is also successfully achieved. Fourth item, as described in chapter 8.1.1, was only partly successful. Measurements could be obtained, but they would be lacking, at the time when this report is written, some essential elements that were targeted according to chapter 2, and also some, which were not even targeted in the trials.

8.1.3 Objective 3

The description of this objective was “Provision of a limited proof-of-concept based on existing hardware”. It included the following subitems:

- Showcase targets: There shall be such a showcase concept that will allow for meaningful and differentiated marketing
 - o Live application showcase

- Target to use TCP-IP to flow control on top of MAC-PHY functions, this together with higher level IP routing (external device) will allow for (un-optimized) IP flow between two terminals
- Co-operation with RRM trials
 - By having co-operating, integrated trials via mutual interfaces, thus replacing 802.xx as a “WINNER emulation” in the RRM trials
- Multi-antenna transmissions: MIMO with modulation and spatial adaptivity

The first subitem “live application showcase” was satisfied with the use of a HDTV camera, the output of which was used for user data, and shown after the receiver in a HDTV-level LCD display. The resolution was always 1920x1080, which is to say, full-HD level. The second subitem “TCP-IP over MAC-PHY” was also successfully implemented, even though many subtleties in carrying TCP-IP over radio link were neglected in the implementation. The third subitem “Co-operation with RRM trials” was also successful, and the two trials were integrated at TCP-IP level, which proved to be quite easy. The last subitem labelled “Multi-antenna transmissions” was only partially successful. The code base was generated to allow spatial adaptivity, but the multi-antenna transmissions were never fully implemented.

9 References

- [And98] R. Andrake, "A Survey of CORDIC algorithms for FPGA based computers", 1998 (<http://www.andraka.com/files/crdcsrvy.pdf>)
- [Aue03] G. Auer, A. Dammann, and S. Sand, "Comparison of low complexity OFDM channel estimation techniques," in Int. OFDM Workshop, Sept. 2003.
- [Auer05] G. Auer, "Efficient Implementation of Robust OFDM Channel Estimation", IEEE 2005
- [CIS+06] R.B. Casey, R. Iyer, M. Saed, B. Nutter, T. Karp, "Blind equalization of an HF channel for a passive listening system", Texas Tech University, Sep 2006
- [Döh91] R Döhler, "IMA Journal of Numerical Analysis", Volume 11, Parts 1-5, January 1991
- [EL06] F. Edman, "Digital Hardware Aspects of Multiantenna Algorithms", Lund 2006
- [Hayes96] Monson H. Hayes, "Statistical digital signal processing and modelling", John Wiley & Sons, 1996
- [Haykin02] Simon Haykin, "Adaptive filter theory", Prentice Hall, 2002
- [HKR1] P. Höher, S. Kaiser, and P. Robertson, "Pilot-symbol-aided channel estimation in time and frequency", in IEEE Global Telecommunications Conference: The Mini-Conference, Phoenix, AZ, Nov. 1997
- [HKR2] P. Hoeher, S. Kaiser and P. Robertson, "Two-dimensional pilot symbol-aided channel estimation by Wiener filtering", Proc. IEEE, ICASSP'97, Munich, April 1997
- [IJ02] Ifeachor E., Jervis B., "Digital Signal Processing: A Practical Approach", 2nd Edition, Prentice Hall 2002. ISBN 0-201-59619-9
- [KCD05] M. Karkooti, J.R. Cavallaro, C. Dick, "FPGA Implementation of Matrix Inversion Using QRD-RLS Algorithm", November 2005
- [RG75] L. R. Rabiner and B. Gold, "Theory and Application of Digital Signal Processing", Englewood Cliffs, NJ: Prentice-Hall, Inc., 1975.
- [RU01] T. J. Richardson and R. Urbanke, "Efficient encoding of low-density parity-check codes," IEEE Transactions on Information Theory, Vol. 47, no. 2, pp 638-656, Feb. 2001.
- [SS00] F. Sanzi, and J. Speidel, "An adaptive two-dimensional channel estimator for wireless OFDM with application to mobile DVB-T", IEEE Transactions on Broadcasting, 46(2):128--133, 2000.
- [SSK06] M. Sternad, T. Svenson, and G. Klang, "WINNER MAC for cellular transmission", in Proc. of IST Mobile Summit, Mykonos, Greece, June 2006
- [The92] Charles W. Therrien, "Discrete random signals and statistical signal processing", Prentice Hall, 1992
- [Vas06] Vaseghi S.V., "Advanced Digital Signal Processing and Noise Reduction", John Wiley & Sons 2006. ISBN 0-470-09494-X
- [WIM] IST-2003-507581 WINNER "D5.1 A set of channel and propagation models for early link and system level simulations", <https://www.istwinner.org/documents/deliverables/d5-1.pdf>, ver 1.0.
- [WIN1D21] IST-2003-507581 WINNER "D2.1 Identification of Radio-link technologies"
- [WIN1D210] IST-2003-507581 WINNER, "D2.10 Final Report on Identified RI Key Technologies, System Concept, and Their Assessment", December 2005.

[WIN1D23]	IST-2003-507581 WINNER, “D2.3 Assessment of Radio-link technologies”
[WIN2D221]	IST-4-027756 WINNER II, “D2.2.1 Joint Modulation and Coding Procedures”
[WIN2D222]	IST-4-027756 WINNER II, “D2.2.2 Link adaptation procedures”, October 2006
[WIN2D6121]	IST-4-027756 WINNER II, “D6.12.1 Detailed WINNER II system trial plan for Phase III trials”, November 2007
[WIN2D6124]	IST-4-027756 WINNER II, “D6.12.4 Limited proof-of-concept of WINNER physical layer with live multimedia application”, November 2007
[WIN2IR6121]	IST-4-027756 WINNER II, “IR6.12.1 Intermediate report on validation and implementation of WINNER II radio architecture”, December 2006
[WIND210]	IST-2003-507581 WINNER “D2.10 v1.0 Final report on identified RI key technologies, system concept, and their assessment”, chapters F3.2-F3.3, December 2005
[Yli06]	J. Ylitalo, “Fixed-beam MIMO scheme,” in 17th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications, Sept. 2006.
[YZhang]	FPGA Implementation of the Flexible Block LDPC Encoder in Winner Project

Appendix A SNR estimation curves

This appendix shows simulation results for a SNR estimation method. The blue color is for average SNR estimate and SNR estimate upper bound curves, and the red color is for estimate error variance and SNR estimate lower bound curves. The first set of results is for non-iterative system, the second set is for a system which tries to iterate better estimates from the control symbols (QPSK), and the third set is for a system which tries to improve from the second system by also iterating from the data symbols (16QAM / 64QAM).

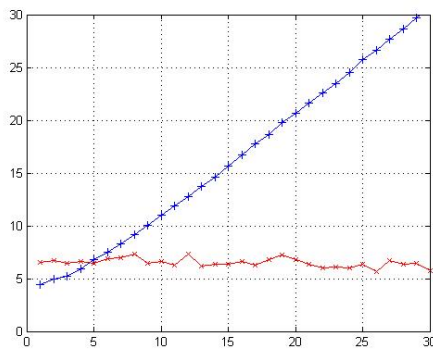


Figure A-1: Average estimate / variance of estimate error from QPSK control symbols, non-iterative

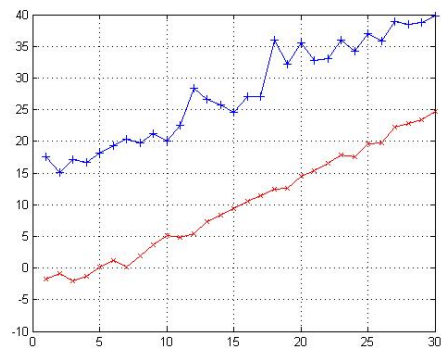


Figure A-2: Maximum / minimum of estimate from QPSK control symbols, non-iterative

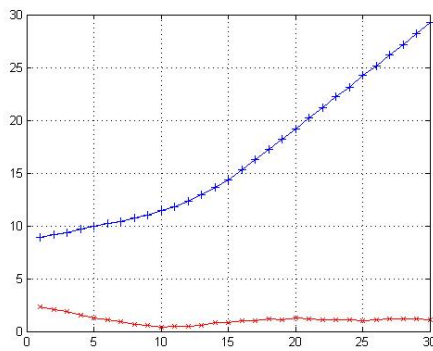


Figure A-3: Average estimate / variance of estimate error from 16QAM data symbols, non-iterative

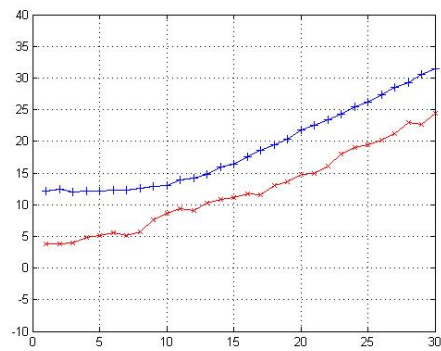


Figure A-4: Maximum / minimum of estimate from 16QAM data symbols, non-iterative

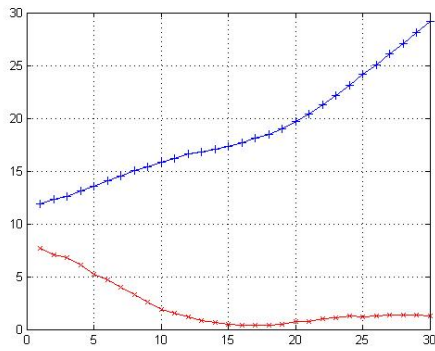


Figure A-5: Average estimate / variance of estimate error from 64QAM data symbols, non-iterative

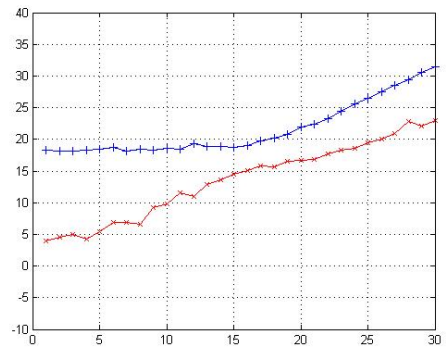


Figure A-6: Maximum / minimum of estimate from 64QAM data symbols, non-iterative

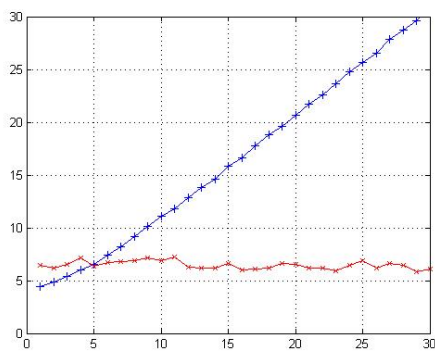


Figure A-7: Average estimate / variance of estimate error from QPSK control symbols, control-iterative

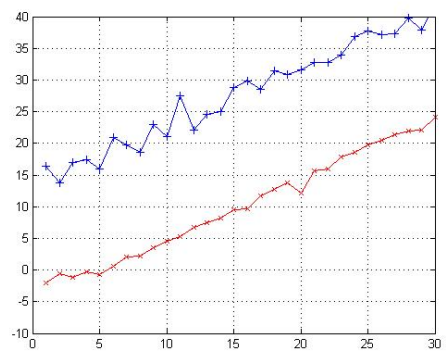


Figure A-8: Maximum / minimum of estimate from QPSK control symbols, control-iterative

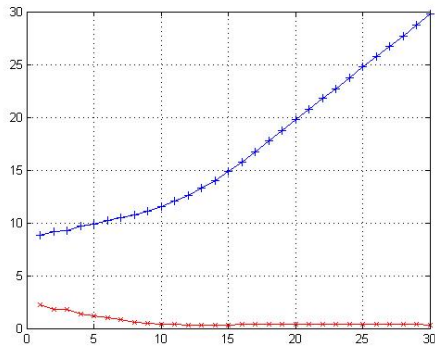


Figure A-9: Average estimate / variance of estimate error from 16QAM data symbols, control-iterative

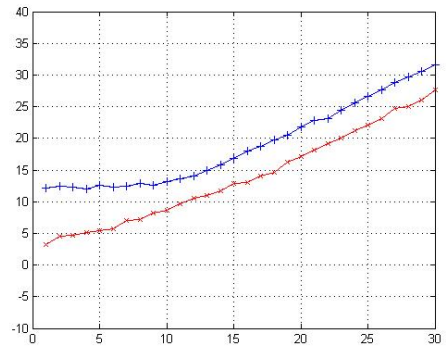


Figure A-10: Maximum / minimum of estimate from 16QAM data symbols, control-iterative

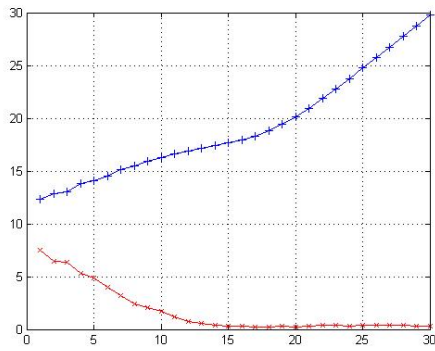


Figure A-11: Average estimate / variance of estimate error from 64QAM data symbols, control -iterative

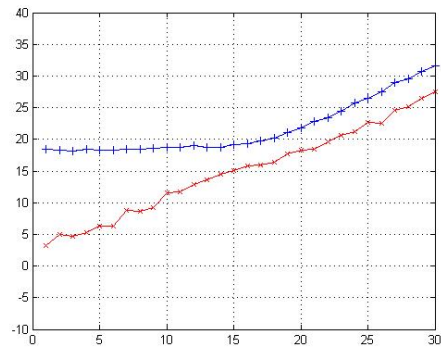


Figure A-12: Maximum / minimum of estimate from 64QAM data symbols, control -iterative

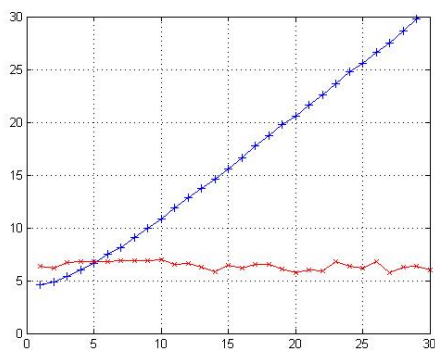


Figure A-13: Average estimate / variance of estimate error from QPSK control symbols, data-iterative

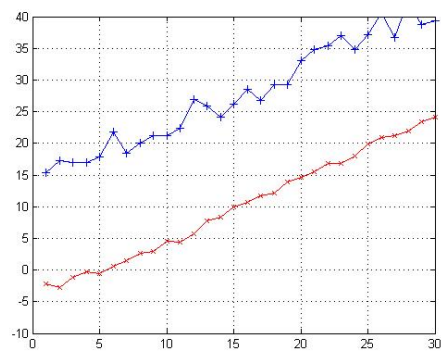


Figure A-14: Maximum / minimum of estimate from QPSK control symbols, data -iterative

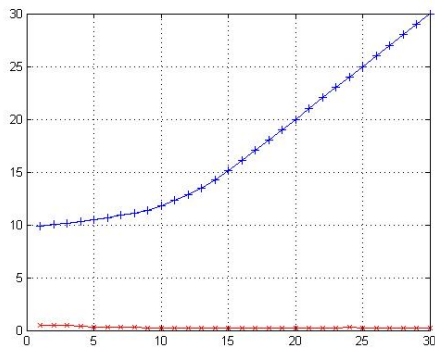


Figure A-15: Average estimate / variance of estimate error from 16QAM data symbols, data -iterative

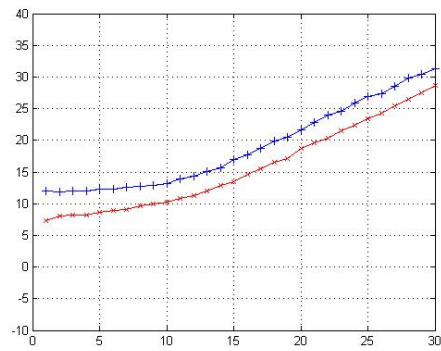


Figure A-16: Maximum / minimum of estimate from 16QAM data symbols, data -iterative

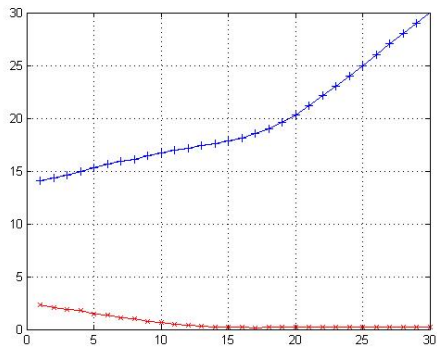


Figure A-17: Average estimate / variance of estimate error from 64QAM data symbols, data -iterative

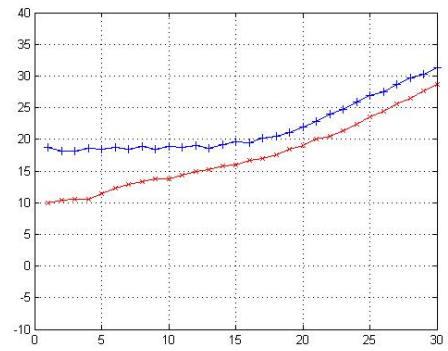


Figure A-18: Maximum / minimum of estimate from 64QAM data symbols, data -iterative